IBM PL/I for MVS & VM
Compiler and Run-Time Migration Guide
Release 1.1

Document Number SC26-3118-01

Note!

Before using this information and the product it supports, be sure to read the general information under “Notices” on page vi.

Second Edition (June 1995)

This edition applies to Version 1 Release 1.1 of IBM PL/I for MVS & VM (named IBM SAA AD/Cycle PL/I MVS & VM for Release 1.0), 5688-235, and to any subsequent releases until otherwise indicated in new editions or technical newsletters. Make sure you are using the correct edition for the level of the product.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

IBM Corporation, Department J58
P.O. Box 49023
San Jose, CA, 95161-9023
United States of America

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Notices

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of IBM may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, NY 10594, U.S.A.

Programming Interface Information

This book is intended to help the customer migrate from previous releases of OS PL/I to PL/I for MVS & VM and Language Environment for MVS & VM. This book documents General-use Programming Interface and Associated Guidance Information provided by PL/I for MVS & VM.

General-use programming interfaces allow the customer to write programs that obtain the services of PL/I for MVS & VM.

Trademarks

The following terms, denoted by an asterisk (*) in this publication, are trademarks of the IBM Corporation in the United States or other countries or both:

AD/Cycle
C/370
CICS
CICS/ESA
COBOL/370
COBOL for MVS & VM
DB2
DFSORT
IBM
IMS/ESA
Language Environment
MVS/DFP
MVS/ESA
OpenEdition
OS/2
SAA
VM/ESA

Chapter 1. Introduction

This book contains information to help you migrate applications from previous releases of OS PL/I to IBM Language Environment for MVS & VM (Language Environment) and IBM PL/I for MVS & VM. It suggests solutions to problems that arise because of differences in support between previous releases of OS PL/I and PL/I for MVS & VM. It explains how to get compatibility support for your applications when running previously compiled OS PL/I programs under Language Environment.

This book also contains brief information about the debugging facilities for PL/I for MVS & VM, IBM Debug Tool and IBM CoOperative Development Environment/370 (CODE/370), and refers to the Debug Tool and CODE/370 publications for more details.

This book is for system programmers, application programmers, and IBM support personnel who are involved in PL/I product migration. Prerequisite knowledge for using this book is:

- A general understanding of your operating system
- Some knowledge of the PL/I language and options
- Some knowledge of how PL/I uses Language Environment for its run-time environment

This books contains major changes in structure and information, so no change bars (|) have been included in the document.

New Names for IBM Products

In 1994, IBM updated the naming-convention requirements for all products so that customers can look at the name and immediately know on what platform a particular product is supported. In order to comply with the new requirements, some products changed their names for a new release even though the version and product number remained the same. Anytime we refer to a product in this book, we will use the new name. To help you know that we are referring to the same product you recall by the old name, we are showing the naming differences in the following table:

Table 1. Name Changes for Products Discussed in this Book

Old Name	New Name
IBM SAA AD/Cycle C/370 (C/370)	n/a
IBM C/C^{++} for MVS/ESA (C/C^{++})	n/a
IBM SAA AD/Cycle COBOL/370, Release 1	IBM COBOL for MVS & VM (COBOL), Release 2
IBM SAA AD/Cycle Language Environment/370, Release 3	IBM Language Environment for MVS & VM, Release 4
IBM SAA AD/Cycle PL/I MVS & VM, Release 1	IBM PL/I for MVS & VM, Release 1.1

Run-time Environment for PL/I for MVS & VM

PL/I for MVS & VM uses Language Environment as its run-time environment. It conforms to Language Environment architecture and shares the run-time environment with other conforming languages such as C/370, C/C++, and COBOL.

Language Environment is a common run-time environment for its conforming language compilers: C/370, C/C++, COBOL and PL/I for MVS & VM. It provides a common set of run-time options and callable services. It also improves interlanguage communication (ILC) between high-level languages (HLL) and assembler by eliminating language-specific initialization and termination on each ILC invocation. Language Environment provides compatibility support for existing applications with a few restrictions that are described in this book.

Debugging Facility for PL/I for MVS & VM

There are two debugging facilities for PL/I for MVS & VM:

- IBM Debug Tool is packaged with PL/I for MVS & VM as an optional feature and is a debugging tool for MVS and VM only.
- CODE/370 has a debugging tool for MVS and VM, but it contains a workstation interface that allows you to edit, compile, and debug from a workstation.

Both debugging facilities provide compatibility support for existing applications with a few restrictions. For more information, read the product books listed in the "Bibliography" on page 58.

PL/I for MVS & VM uses the IBM Debug Tool as its debugging facility on MVS and VM. Debug Tool utilizes the common run-time environment, Language Environment, to provide ILC debugging capability among C/370, C/C++, COBOL, and PL/I for MVS & VM, and supports PL/I multitasking. It also provides debugging capability under CICS. Debug Tool is compatible with INSPECT for C/370 and PL/I and the OS PL/I interactive test facility, PLITEST. Its features are equivalent to the features PLITEST supports for OS PL/I. Debug Tool provides compatibility support for OS PL/I Version 2 applications and the same level of toleration that PLITEST used to provide for OS PL/I Version 1 applications.

CODE/370 supports editing, compiling, and debugging host (MVS or VM) programs from a workstation. It uses a language-sensitive editor and debugging tool that allows you to edit, compile, and debug your programs residing on the host while you are using your workstation. This gives you the opportunity to download some of the host development workload.

Using Your Documentation

The publications provided with PL/I for MVS & VM are designed to help you do PL/I programming under MVS or VM. Each publication helps you perform a different task.

Where to Look for More Information

The following tables show you how to use the publications you receive with PL/I for MVS & VM and Language Environment. You'll want to know information about both your compiler and run-time environment. For the complete titles and order numbers of these and other related publications, such as the IBM Debug Tool, see the “Bibliography” on page 58.

PL/I Information

Table 2. How to Use Publications You Receive with PL/I for MVS & VM

To...	Use...
Understand warranty information	Licensed Programming Specifications
Plan for, install, customize, and maintain PL/I	Installation and Customization under MVS Program Directory under VM
Understand compiler and run-time changes and adapt programs to PL/I and Language Environment	Compiler and Run-Time Migration Guide
Prepare and test your programs and get details on compiler options	Programming Guide
Get details on PL/I syntax and specifications of language elements	Language Reference Reference Summary
Diagnose compiler problems and report them to IBM	Diagnosis Guide
Get details on compile-time messages	Compile-Time Messages and Codes

Language Environment Information

Table 3. How to Use Publications You Receive with Language Environment for MVS & VM

To...	Use...
Evaluate Language Environment	Fact Sheet Concepts Guide
Understand warranty information	Licensed Program Specifications
Understand the Language Environment program models and concepts	Concepts Guide Programming Guide
Plan for, install, customize, and maintain Language Environment on MVS	Installation and Customization under MVS Program Directory under VM
Migrate applications to Language Environment	Run-Time Migration Guide Your language migration guide
Find syntax for run-time options and callable services	Programming Reference
Develop your Language Environment-conforming applications	Programming Guide and your language programming guide
Find syntax for run-time options and callable services	Programming Reference
Develop interlanguage communication (ILC) applications	Writing Interlanguage Communication Applications
Debug your Language Environment-conforming application and get details on run-time messages	Debugging Guide and Run-Time Messages
Diagnose problems with Language Environment	Debugging Guide and Run-Time Messages
Find information in the Language Environment library quickly	Master Index

Chapter 2. Considerations Before Migrating

Before you migrate to Language Environment or PL/I for MVS & VM, you should read this chapter. It discusses the functional differences between OS PL/I and PL/I for MVS & VM (and its run-time environment Language Environment). These differences should be considered before you install Language Environment or PL/I for MVS & VM. Other chapters in this book discuss differences you must consider during and after installation.

It is possible to install Language Environment together with the OS PL/I library. If you'd like to know more about this topic, see Chapter 8, “OS PL/I Coexistence with Language Environment” on page 50.

This chapter includes the following sections:

- Differences in PLICALLA and PLICALLB support
- Differences in preinitialization support
- Differences in PLISRTx support
- Differences in multitasking support
- Differences in DATE/TIME built-in functions
- Differences in user return codes
- Differences in condition handling
- Differences in run-time messages
- Differences in PLIDUMP
- Differences in debugging tools
- Differences in run-time options
- Differences in storage report
- Differences in interlanguage communication support
- Differences in assembler support
- Considerations for better CPU and storage utilization

Differences in PLICALLA and PLICALLB Support

The interfaces in the following sections are not recommended for use in PL/I for MVS & VM. They are only supported for compatibility reasons.

PLICALLA Considerations

Language Environment provides support for OS PL/I applications that use the PLICALLA entry point. It also provides support for recompiled OS PL/I applications that want to continue to use PLICALLA as the primary entry point. However, if you recompile every OS PL/I program in the main load module with PL/I for MVS & VM, you must do one of the following when you link your main load module:

- For MVS applications, concatenate SIBMCALL before SCEELKED;
  for VM applications, global SIBMCALL before SCEELKED.
- Explicitly INCLUDE Language Environment-provided PLISTART CSECT.

If you don't do this, the linkage editor or loader issues an error message for an unresolved ENTRY PLICALLA statement.

Note that the above rule of using SIBMCALL or explicitly including PLISTART CSECT does not apply if any of the following conditions exist:

- At least one OS PL/I program in the main load module is not recompiled with PL/I for MVS & VM. In this case the PLICALLA label entry in the OS PL/I object module is used to resolve the linkage editor ENTRY PLICALLA control statement.
- The OS PL/I main load module is statically linked with an assembler program that invokes PLICALLA. In this case, the V(PLICALLA) in the assembler program will automatically use the Language Environment-provided PLISTART CSECT to resolve the linkage editor ENTRY PLICALLA control statement.

You can also use PLICALLA as the primary entry point of a FETCHed/CALLed main load module compiled with either OS PL/I or PL/I for MVS & VM. However, the calling routine must pass only user arguments which are passed as to a subroutine. If run-time options are passed, they are treated as user arguments.

If you have a multitasking application, SIBMTASK must be concatenated in front of SCEELKED. SIBMCALL and SIBMTASK can appear in any order. For more details, see "Differences in Multitasking Support" on page 10.

You should not place SIBMCALL in front of SCEELKED or explicitly INCLUDE Language Environment-provided PLISTART CSECT for a load module not using PLICALLA because it will increase the size of the module.

If you develop a new application in PL/I for MVS & VM or OS PL/I and you want the main procedure to receive user arguments like a subroutine, you should do one of the following:

- Receive control directly from IMS by
  - Using CEESTART or PLISTART as the primary entry point of the load module
  - Specifying the SYSTEM(IMS) compile-time option
- Receive control from an assembler program or a procedure using a FETCH or CALL statement by:
  - Using CEESTART or PLISTART as the primary entry point of the load module
  - Specifying the NOEXECOPS option and the SYSTEM(MVS) compile-time option
  - Specifying either BYADDR or BYVALUE option. Refer to the *Language Reference* for a description of these options.

Language Environment support of PLICALLA is not available in the following environments:

CICS environment
Preinitialized environment
Nested enclave environment except the PL/I FETCHable main.

Passing Parameters

OPTIONS(BYADDR) passes the argument indirectly by reference or value and is the usual argument-passing convention. PL/I for MVS & VM also provides OPTIONS(BYVALUE) which passes arguments directly by value.

All OS PL/I applications invoking PLICALLA use the BYADDR convention to receive arguments.

If the main procedure is recompiled with PL/I for MVS & VM using SYSTEM(CICS|IMS), only POINTER data type can be passed as parameters. If a main procedure receives control from assembler via PLICALLA and is recompiled with PL/I for MVS & VM, the main procedure cannot be compiled with SYSTEM(CICS|IMS).

Table 4 provides the expected argument passing convention (either BYADDR or BYVALUE) when the main procedure of your OS PL/I PLICALLA application is recompiled with PL/I for MVS & VM:

Table 4. Parameter Passing for the Main Procedure Compiled with PL/I for MVS & VM

System environment	Invoked from IMS[1]	Invoked from assembler program[2]	Invoked by PL/I FETCH/CALL Statement[2]
SYSTEM(MVS)	BYADDR	BYADDR	BYADDR
SYSTEM(CMS|CMSTPL)	BYADDR	BYADDR	BYADDR
SYSTEM(CICS)	BYVALUE[3]	Not supported	Not supported
SYSTEM(IMS)	BYVALUE[3]	Not supported	Not supported
SYSTEM(TSO)	BYADDR	BYADDR	BYADDR

[1]LANG=PL/I must be specified and it passes indirect by reference.
[2]It must have already passed indirect by reference or by value.
[3]PL/I library will convert the argument list to direct by value.

PLICALLB Considerations

Language Environment provides support for OS PL/I applications that use the PLICALLB entry point. It also provides support for recompiled OS PL/I PLICALLB applications that want to continue to use PLICALLB as the primary entry point. The following table shows the PLICALLB parameter mapping between OS PL/I and Language Environment:

Table 5 (Page 1 of 2). Differences in Support of PLICALLB Argument List

OS PL/I	Language Environment for MVS & VM
Address of argument list (argument must either point to an address or be zero)	Same support
Address of the length of ISA storage for a nonmultitasking program or the major task in a multitasking program	Mapped to STACK(*init_size*)
Address of ISA storage	Used as the initial STACK segment
Address of the length of ISA storage for each subtask	Mapped to NONIPTSTACK(*init_size*)
Address of the maximum number of concurrent subtasks	Mapped to PLITASKCOUNT(*max_thread*)

Table 5 (Page 2 of 2). Differences in Support of PLICALLB Argument List

OS PL/I	Language Environment for MVS & VM
Address of the options word, in which the following run-time options can be specified:	Supported as follows:
REPORT	REPORT mapped to RPTSTG
SPIE|STAE	SPIE|STAE mapped to TRAP
COUNT	COUNT ignored
FLOW	FLOW ignored
HEAP suboptions	HEAP(,,KEEP|FREE)|(,,ANY|BELOW)
TASKHEAP suboptions	THREADHEAP(,,KEEP|FREE)|(,,ANY|BELOW)
Address of HEAP storage length for a nonmultitasking program or the major task in a multitasking program	Mapped to HEAP(*init_size*)
Address of HEAP storage	Used as the initial HEAP segment
Address of HEAP increment for a nonmultitasking program or the major task in a multitasking program	Mapped to HEAP(,*incr_size*)
Address of HEAP for subtasks	Mapped to THREADHEAP(,*increment*)
Address of ISA increment for a nonmultitasking program or the major task in a multitasking program	Mapped to STACK(,*incr_size*)
Address of ISA increment for each subtask (optional for a nontasking application)	Mapped to NONIPTSTACK(,*incr_size*)

When the above argument list is passed in via the PLICALLB entry point, the argument in the list must either point to an address or be zero. The high-order bit ON in an argument indicates the end of the argument list. R1 must contain the address of the argument list.

With Language Environment, the run-time options passed via the PLICALLB entry point are processed as options specified on invocation of the application and have a higher precedence than CEEUOPT or PLIXOPT options. The assembler user exit cannot be used to alter the run-time options passed through the PLICALLB invocation. To summarize, the run-time options passed in have the following precedence (from highest to lowest) among Language Environment option specification methods:

1. Options defined at installation time that have the non-overrideable attribute
2. Options specified via the PLICALLB entry point
3. Options specified in the PLIXOPT string or in CEEUOPT
4. Option defaults defined at installation time

The user arguments passed to the PL/I main routine have the following precedence (from highest to lowest):

1. Output from CXIT_PARM or AUE_PARM of the assembler user exit
2. User arguments passed in via the PLICALLB entry

Note: The input to CXIT_PARM or AUE_PARM of the assembler user exit is the first argument in the PLICALLB parameter list, that is, the address of a vector of user argument addresses.

Language Environment encourages the use of above-16M-line storage. For compatibility with OS PL/I, Language Environment maps the user-supplied ISA and

HEAP storage to STACK and HEAP. With this mapping, however, you must be aware that Language Environment still needs to issue some GETMAINs. Since user-supplied ISA/HEAP storage is usually below the 16M line. below-16M-line storage can be quickly consumed under Language Environment. How Language Environment manages storage can be found in the *Language Environment for MVS & VM Programming Guide*.

Language Environment manages storage differently than OS PL/I. It divides storage into more categories than the two OS PL/I supported, ISA and HEAP. As a result, mapping the user-supplied OS PL/I ISA or HEAP storage to Language Environment STACK or HEAP storage still requires GETMAINs during run time. Further, Language Environment provides diagnostics to ensure the user-supplied length of ISA or HEAP storage is a multiple of 8 bytes and the address is on a double-word boundary.

Language Environment also ensures the location of the user-supplied ISA or HEAP storage matches to the location specification in the STACK or HEAP run-time option. When the user-supplied HEAP storage is ignored because of the following reasons:

1. User-supplied heap storage is above the 16M line, and
2. The ANYWHERE suboption of the HEAP option is in effect, and
3. The main program is in AMODE(24).

Language Environment allocates below the 16M line storage using the init_sz24 and incr_sz24 suboptions specified in the HEAP option.

If you recompile every program in the main load module with PL/I for MVS & VM and you want to continue to use PLICALLB as the primary entry point, you must do one of the following when you link your main load module:

- For MVS applications, concatenate SIBMCALL before SCEELKED;
  for VM applications, global SIBMCALL before SCEELKED.
- Explicitly INCLUDE Language Environment-provided PLISTART CSECT.

Note that the above rule of using SIBMCALL or explicitly including PLISTART CSECT does not apply if any of the following conditions exist:

- At least one OS PL/I program in the main load module is not recompiled with PL/I for MVS & VM. In this case the PLICALLB label entry in the OS PL/I object module is used to resolve the linkage editor ENTRY PLICALLB control statement.
- OS PL/I main load module is statically linked with an assembler program that invokes PLICALLB. In this case, the V(PLICALLB) in the assembler program will automatically use the Language Environment-provided PLISTART CSECT to resolve the linkage editor ENTRY PLICALLB control statement.

You should not place SIBMCALL in front of SCEELKED or explicitly INCLUDE Language Environment-provided PLISTART CSECT for a load module not using PLICALLB because it will increase the size of the module.

When you develop new applications in PL/I for MVS & VM and want to pass both run-time options and arguments to a PL/I main procedure, especially to provide user-supplied stack and heap storage from an assembler program, take advantage

of Language Environment's preinitialization support as described in *Language Environment for MVS & VM Programming Guide*.

Language Environment support of PLICALLB is not available in the following environments:

CICS
IMS
Preinitialized environment
Nested enclave environment

Passing Parameters

OPTIONS(BYADDR) passes the argument indirectly by reference or value and is the usual argument-passing convention. PL/I for MVS & VM also provides OPTIONS(BYVALUE) which passes arguments directly by value.

You must use the BYADDR option when you want to pass parameters using PLICALLB. PLICALLB is invoked from assembler which passes the argument list indirectly by reference or value.

Table 6 provides the expected argument passing convention (either BYADDR or BYVALUE) when the main procedure of your OS PL/I PLICALLB application is recompiled with PL/I for MVS & VM:

Table 6. Parameter Passing for the Main Procedure Compiled with PL/I for MVS & VM

System environment	Invoked from assembler program[1]
SYSTEM(MVS)	BYADDR
SYSTEM(CMS|CMSTPL)	BYADDR
SYSTEM(CICS)	Not supported
SYSTEM(IMS)	Not supported
SYSTEM(TSO)	BYADDR

[1]It passed the argument list required by the PLICALLB entry.

Differences in Preinitialization Support

The PL/I preinitialized program interface is supported with the following changes:

- The PL/I preinitialized program interface no longer supports the REINITIALIZE request modifier code. If you attempt to use this function, it is diagnosed with the 4093-136 abend code.
- If the routine specified in the CALL request is not statically linked with the assembler driver and it contains ILC, you must ensure the environment for ILC is initialized by including the same ILC in the routine specified in the INIT request.
- The TERM request no longer returns 1000 return code as OS PL/I run time did.
- Some of the return and reason codes for the service vector defined by OS PL/I are changed. You must use the return and reason codes for the service vector defined by Language Environment preinitialization services as described in *Language Environment for MVS & VM Programming Reference*.

- If you are using VM, the LOAD and DELETE services must have the additional searching capability to include the saved segment and relocatable load modules.

Language Environment preinitialization services support multiple preinitialization environments under the same TCB. Multiple preinitialization environments under the same TCB is not supported by OS PL/I. To understand how the service works, see "Using Preinitialization Services" in *Language Environment for MVS & VM Programming Guide*.

Differences in PLISRTx Support

OS PL/I applications containing PLISRTx invocations are supported by Language Environment. However, it is a good idea to relink your load module with Language Environment for the following reasons:

- To allow the library routine to access the Language Environment-provided DFSORT interface for a more integrated language and sort environment.
- To allow the library routine to replace the 24-bit DFSORT parameter list with the extended 31-bit DFSORT parameter list.

You can relink your OS PL/I PLISRTx applications using one of the following ways:

- Object module relinking tool on OS PL/I Version 2 Release 3

  See "OS PL/I Object Module Relinking Tool - APARs PN69803" on page 56 for details.
- Library routine replacement tool on Language Environment

  See "OS PL/I Library Routine Replacement Tool" on page 53 for details.
- Relink the object module directly with Language Environment.

Differences in Multitasking Support

The syntax and semantics of PL/I multitasking facility is supported by PL/I for MVS & VM and Language Environment for MVS & VM the same way as it was by OS PL/I. However, Language Environment uses different underlying system services to support PL/I multitasking.

For PL/I functions which are common to nontasking and multitasking, their migration considerations described in this book apply to both nontasking and multitasking applications. Thus, if your multitasking application uses a specific common function, such as PLICALLA, and you want to know if there is any migration considerations, you should find and read the discussion for that function in this book.

Language Environment has the following unique requirements and differences from OS PL/I for PL/I multitasking applications. You must follow the requirements and observe the differences when you run your multitasking applications.

- MVS/ESA SP Version 5 with OpenEdition (OE) MVS Services is required; otherwise you will receive message IBM058I and 4093-52 abend.
- Language Environment explicitly diagnoses the environments that do not support PL/I multitasking with the message IBM0577I and 4093-12 abend.

Those environments include CICS, DB2, IMS, VM, preinitialized environment, and nested enclaves.

- Language Environment uses different system services to support PL/I multitasking. As a result, each task might experience a timing difference on task initialization and termination. Instead of relying on system services and default tasking hierarchy, you should use the EVENT option or variable and the WAIT statement to control the desired sequence among tasks.
- When you link your PL/I for MVS & VM or relink your OS PL/I multitasking application with Language Environment, you must concatenate the Language Environment library SIBMTASK before SCEELKED for the main load module. Such concatenation is not required for a fetched load module.
- When the SIBMCALL dataset is used to support OS PL/I PLICALLA or PLICALLB entry point, the concatenation sequence between SIBMCALL and SIBMTASK can be in any order.
- When the SIBMMATH dataset is used for the OS PL/I math library, the concatenation sequence between SIBMMATH and SIBMTASK can be in any order.
- OS PL/I Version 1 Release 5.1 multitasking load module is supported.
- The OS PL/I multitasking run-time options are mapped to the equivalent Language Environment options as shown in the following table:

Table 7. OS PL/I and Language Environment Run-Time Options Comparison

OS PL/I	Language Environment	Notes
ISASIZE(init_size, sub_task_init_size, max_tasks)	STACK(init_size) NONIPTSTACK (sub_task_init_size) PLITASKCOUNT (max_tasks)	This OS PL/I option is mapped to three Language Environment options. The abbreviation ISA is supported by Language Environment
ISAINC(incr_size, sub_task_incr_size)	STACK(,incr_size) NONIPTSTACK (,sub_task_incr_size)	This OS PL/I option is mapped to two Language Environment options.
TASKHEAD (sub_task_init_size, sub_task_incr_size, loc, state)	THREADHEAP (sub_task_init_size, sub_task_incr_size, loc, state)	These options map directly. The abbreviation TH will also be recognized for TASKHEAP.

- Some of Language Environment run-time options have specific meanings for PL/I multitasking applications. For example, POSIX(OFF) IBM-supplied default must be used with PL/I multitasking application; otherwise the message IBM0581I and abend 4093-52 are issued. See *Language Environment for MVS & VM Programming Reference* for details.
- When you use the PRIORITY option on the CALL statement to create a subtask with a different priority, that priority does not take effect until the subtask is dispatched a second time by the system. When the subtask is created, it always inherits the priority of the creating task the first time it is dispatched by the system. Only when the subtask is dispatched by the system a second time does the different priority take effect. However, if you use the PRIORITY built-in function or pseudovariable to reflect a different priority in the subtask, the different priority of the subtask takes effect immediately.

- When SYSPRINT is shared by multiple tasks, Language Environment no longer uses MVS ENQ and DEQ to serialize the stream PUT operations but MVS ENQ and DEQ remain in use for an EXCLUSIVE file.

  If you want the PL/I standard SYSPRINT file to contain run-time message output as well as the usual PL/I STREAM output, you should specify the MSGFILE(SYSPRINT) run-time option. In addition, the SYSPRINT file must be opened in the major task before any subtasks are created; otherwise, Language Environment raises the UNDEFINEDFILE condition along with message IBM0580S. OS PL/I does not diagnose this rule.

  When MSGFILE(SYSPRINT) is in effect, the user output lines for an individual stream PUT statement may be interwoven with run-time message output from other tasks. This differs from the output for the OS PL/I diagnostic SYSPRINT file under OS PL/I multitasking environment.

- If a file is closed in a task, results are unpredictable if an attempt is made to utilize the file in any other task which had previously shared the file.

- DFSORT/MVS Release 13 provides the cultural sort support via the LOCALE parameter. The DFSORT cultural sorting is not supported for a multitasking application.

  If your application is OS PL/I, you must make sure DFSORT/MVS Release 13 is installed with LOCALE=NONE.

  If your application is PL/I for MVS & VM and is relinked with Language Environment, or is relinked with the OS PL/I Object Module Relinking tool on OS PL/I Version 2 Release 3 (APAR PN69803 and PN69804), you don't need to be concerned with the LOCALE parameter. Language Environment will enforce LOCALE=NONE for your application.

- Language Environment provides equivalent multitasking information in PLIDUMP as in OS PL/I.

- Language Environment provides equivalent multitasking information in the storage report as in OS PL/I.

- Language Environment uses *thread* in some run-time outputs such as run-time messages and storage report when referring to a PL/I task.

- OS PL/I Version 2 Release 3 Programming Guide, page 88, lists certain library modules that must be INCLUDEd in the main task while the fetched subtask uses certain tasking functions. Such INCLUDE is no longer required when you relink your OS PL/I application with Language Environment.

- Interlanguage Communication with COBOL

  Interlanguage communication (ILC) with COBOL remains supported. Language Environment provides a better enforcement of the rule that, if a COBOL program has been invoked in a task, no COBOL program can be invoked in other tasks until the task that has invoked COBOL terminates.

- Assembler programs remain supported. However, you must not use the MVS ATTACH and DETACH macros in an assembler program. There are other restrictions on using MVS macros, for example WAIT and POST. See *Language Environment for MVS & VM Programming Guide* for information on these restrictions.

Differences in DATE/TIME Built-In Functions

The DATETIME and TIME built-in functions now return the number of milliseconds in all environments. The syntax and description of these built-in functions are in *PL/I for MVS & VM Language Reference*.

Differences in User Return Code

PL/I for MVS & VM and Language Environment support a FIXED BIN(31) four-byte user return code value for PLIRETC, PLIRETV, and OPTIONS(RETCODE). This support removes the restriction of maximum value 999. OS PL/I applications must be relinked with Language Environment or recompiled with PL/I for MVS & VM in order to take advantage of the four-byte user return-code value.

The following table shows how PL/I user return code is supported:

Table 8. Return Code Behavior under Language Environment for MVS & VM

Function	OS PL/I load module	OS PL/I object module linked with Language Environment	PL/I for MVS & VM load module
PLIRETC built-in function	2-byte value with restriction of 999	4-byte value without restriction of 999	4-byte value without restriction of 999
PLIRETV built-in function	2-byte value	Lower 2 bytes of a 4-byte value	4-byte value
RETCODE option	Lower 2 bytes of R15	Lower 2 bytes of R15	2-byte value

For PLIRETC, the PL/I for MVS & VM and relinked OS PL/I load modules can set a 4-byte user return code value.

For PLIRETV and RETCODE, only the PL/I for MVS & VM load module can receive a 4-byte user return code value.

In order to fully exploit the 4-byte user return code value, you must compile your application with PL/I for MVS & VM.

Under Language Environment, upon return from the PLISRTx invocation, the PL/I user return code is always reset to zero. This is not the case previously with OS PL/I run-time.

Differences in Condition Handling

PL/I condition handling semantics remain supported under Language Environment. However, the timing of issuing the run-time message for an ERROR condition with respect to the ERROR ON-Unit is slightly different in the following way:

- The run-time message for an ERROR condition is issued only if there is no ERROR ON-Unit established, or if the ERROR ON-Unit does not recover from the condition by using a GOTO out of block. Therefore, you can use a GOTO out of the ERROR ON-Unit to avoid a message for a PL/I ERROR condition.

Notice for PL/I conditions whose implicit action including issuing a message and raising the ERROR condition, the timing of issuing the message is unchanged.

Table 9 shows when the run-time message for an ERROR condition is issued under OS PL/I with respect to the ERROR On-Unit.

Table 9. OS PL/I Version 2 Release 3 ERROR ON-Unit and Message for an ERROR condition

Condition	No ON-Units	ERROR ON-Unit No GOTO	ERROR ON-Unit GOTO
ERROR condition raised[1]	Message	Message prior to ON-unit	Message prior to ON-unit
ZERODIVIDE condition raised[2]	Message	Message prior to ON-unit	Message prior to ON-unit

1 Taking the square root of a negative number, data exception, etc.

2 With no ZERODIVIDE ON-unit; thus, implicit action is taken. Message is printed, ERROR condition is raised.

Table 10 shows when the run-time message for an ERROR condition is issued under Language Environment with respect to the ERROR On-Unit.

Table 10. Language Environment ERROR ON-Unit and Message for an ERROR Condition

Condition	No ON-units	ERROR ON-unit No GOTO	ERROR ON-unit GOTO
ERROR condition raised[1]	Message	Message after ON-unit	No message
ZERODIVIDE condition raised[2]	Message	Message prior to ON-unit	Message prior to ON-unit

1 Taking the square root of a negative number, data exception, etc.

2 With no ZERODIVIDE ON-unit; thus, implicit action is taken. Message is printed, ERROR condition is raised.

The SNAP traceback message produced by ON ERROR SNAP continues to be issued before the ERROR ON-unit receives control. Notice the SNAP traceback message is not identical to the regular ERROR message.

Severities of some PL/I conditions are different under Language Environment. See *PL/I for MVS & VM Language Reference* for those severities.

If your OS PL/I application used to force an abend for an unhandled condition under OS PL/I run-time using OS PL/I assembler user exit IBMBXITA or abend exit IBMBEER, you now should use the following ways to force an abend under Language Environment:

- Run your application with Language Environment ABTERMENC(ABEND) option. You cannot specify your own abend code via the run-time option.
- Use Language Environment assembler user exit CEEBXITA to force an abend with your own abend code.

There is limited support for OS PL/I IBMBXITA and IBMBEER under Language Environment. See “Considerations for Using Assembler User Exits” on page 26 for details.

An UNHANDLED condition of severity 2 or higher now produces an abend U4039 and optionally a system dump if SYSUDUMP or SYSABEND ddname is present. If ABTERMENC(RETCODE) is in effect, your application continues the termination with an abend code. If you don't want to see the U4039 abend, Language Environment provides you the facilities to suppress it.

See “Abnormal Termination Exit” in *Language Environment for MVS & VM Installation and Customization under MVS* for ways to suppress or change the U4029 abend.

Differences in Run-Time Messages

The format and content of run-time messages are different. If you have applications that analyze run-time messages, you must change the applications to allow for the differences. The differences include:

- The message number in the message prefix is four digits instead of three digits in the form IBM*nnnnx*, where *nnnn* represents the message number and *x* represents the severity of the message.
- The message severity in the message prefix can be I, W, E, S, or C.
- The message text of some mixed-case English and Japanese messages has been enhanced. However, the message text of uppercase English messages has not changed.

Details are provided in *Language Environment for MVS & VM Debugging Guide and Run-Time Messages*.

Under Language Environment, run-time messages go to the MSGFILE destination specified in the run-time option MSGFILE. The default for MSGFILE destination is SYSOUT. The user output still goes to SYSPRINT. If you want your run-time messages to go to SYSPRINT, specify the MSGFILE(SYSPRINT) run-time option. In this case, SYSPRINT contains user output and run-time messages. Details are provided in the *Language Environment for MVS & VM Programming Guide*.

Under Language Environment, run-time messages give offset values that are relative to the start of the external procedure, rather than relative to the start of the block that contains the statement. You can use these offsets to help you find the statement that is in error. To do this, match the offset provided in the message with the offset given in the pseudo-assembler listing that the compiler produces when you specify the LIST compile-time option.

Differences in PLIDUMP

PLIDUMP now produces a Language Environment-style dump. The way you use PLIDUMP and the dump output is different. The following list the differences in the way you use PLIDUMP and the output produced. *Compile unit* refers to the primary entry point of the external procedure and *Compile unit name* refers to the name of the external procedure.

- The ddname of the dump output file can be CEEDUMP, PLIDUMP, or PL1DUMP. If you do not define one of these files, Language Environment for MVS & VM creates a default CEEDUMP file to contain the dump output. The LRECL of the dump output file must be at least 133 bytes to prevent dump records from wrapping, not the 121 bytes required by OS PL/I.

- When you use the hexadecimal (H) option of PLIDUMP, you must specify the ddname CEESNAP for MVS, or the file name CEESNAP for VM; otherwise the H option is ignored. This data set contains the SNAP dump output.

  When you specify the hexadecimal (H) option under MVS, the output from SNAP includes all system control program information (SDATA=ALL). OS PL/I provides only partial information (SDATA=CB, Q, and TRT).
- When you use ILC, the dump output contains information related to other languages (for example, C/C++ or COBOL).
- The identifier character string is limited to 60 bytes rather than the 90 bytes OS PL/I supported.
- The traceback section lists the compile-unit name associated with each entry point name. When the entry point is a secondary entry point, the primary entry point name associated with the actual entry point is not listed.

  The traceback section also contains offsets relative to the address of the compile unit, as well as offsets relative to the address of the real entry point.
- Run-time messages are in a separate section; they are no longer part of the traceback section.
- When you specify the Block (B) option of PLIDUMP, the condition handler save areas appear in the Block section of the dump. If you do not specify the Block option of PLIDUMP, the condition handler save areas do not appear in the dump.
- If the program was compiled with the TEST compile-time option, and a begin-block has a label, the begin-block is identified as `Label:BEGIN block.`. Otherwise, the begin-block is identified as `%BLOCK`*nn*`,` where *nn* is the block count for the begin-block.
- Compiler-generated ILC subroutines now appear in the traceback section. They are identified as the compile unit name concatenated with the suffix ILC.
- PL/I library routines that have Language Environment-defined Program Prologue Areas (PPAs) are identified by name in the dump. If the library routines do not have Language Environment PPAs, they are identified as `Library(PL/I).`
- Assembler routines that conform to the rules for mimicking PL/I routines are identified by their CSECT names in the dump output.
- PLIDUMP now conforms to National Language Support standards.
- PLIDUMP can supply information across multiple Language Environment enclaves. For example, if an application running in one enclave FETCHes a main procedure (an action that creates another enclave), PLIDUMP contains information about both procedures.

Differences between Debugging Tools

IBM Debug Tool is the interactive debugger that supports PL/I and Language Environment. Debug Tool functions are equivalent to PLITEST functions. However, some names of PLITEST commands have changed in Debug Tool and are no longer accepted. These are listed in Table 11.

You must have Language Environment for MVS & VM Release 4 installed on your system before you can use Debug Tool with your PL/I for MVS & VM or OS PL/I applications.

Table 11. PLITEST Commands and Their Debug Tool Equivalents

PLITEST Command	Equivalent Debug Tool Command
CLEAR ON	CLEAR AT OCCURENCE
LIST %FPRS	LIST SHORT FLOATING
LIST %LPRS	LIST LONG FLOATING
LIST %GPRS	LIST REGISTERS
LIST SNAP	LIST CALLS
MOVECURS	CURSER
ON	AT OCCURENCE
QUERY AT	LIST AT
QUERY ATTRIBUTES	DESCRIBE ATTRIBUTES
QUERY BEARINGS	QUERY LOCATION
QUERY ENVIRONMENT	DESCRIBE ENVIRONMENT
QUERY MONITOR	LIST MONITOR
QUERY NAMES 'pattern'	LIST NAMES 'pattern'
QUERY NAMES PROCEDURE	LIST PROCEDURE
QUERY PROGRAM	DESCRIBE PROGRAM
QUERY STATEMENT NUMBERS	LIST STATEMENT NUMBERS
SEARCH	FIND
SET GRAPHIC	SET DBCS
SET LANGUAGE	SET NATIONAL LANGUAGE
SET LAST n	SET HISTORY n
SET FILE	SET LOG
SIGNAL (ON cond) PROGRAM	TRIGGER (ON cond)
SIGNAL (ON cond) TEST	TRIGGER AT OCCURENCE (ON cond)
SIGNAL (AT cond) TEST	TRIGGER AT (AT cond)
VTRACE	STEP
WINDOWS	LAYOUT

Differences in Run-Time Options

Language Environment run-time options replace PL/I run-time options. Most PL/I run-time options have an equivalent Language Environment run-time option that provides the same function. This section describes differences in the use of run-time options.

You should adapt your applications to allow for the following differences:

- When you pass run-time options in the MVS GO step, your run-time options string must end with a slash (/) to distinguish it from a main procedure parameter string. If you omit the slash, the string is passed as the main procedure parameter.
- The Language Environment ABTERMENC option controls which type of return/abend code your application receives at abnormal termination.

ABTERMINC(RETCODE) allows your application to receive run-time return code, which is equivalent to the way OS PL/I worked.

- The OS PL/I COUNT option is ignored.
- The Language Environment ERRCOUNT option limits the number of errors that are handled at run time. ERRCOUNT(0) specifies that there is no limit, which is equivalent to the way the OS PL/I worked.
- The Language Environment DEPTHCONDLMT option limits the extent to which conditions can be nested. To maintain compatibility, specify DEPTHCONDLMT(0), which means there is an unlimited depth.
- The OS PL/I FLOW option is ignored.
- The OS PL/I HEAP option is always in effect. This means that when you allocate storage for BASED and CONTROLLED variables, the storage always comes from HEAP storage. The storage does not come from a PL/I Initial Storage Area (ISA). HEAP(0) is ignored and not supported.
- The Language Environment NATLANG option replaces the OS PL/I LANGUAGE option.
- The Language Environment RPTSTG option replaces the OS PL/I REPORT option and the option report.
- The Language Environment TRAP option replaces both OS PL/I SPIE and STAE options. The following table shows how the OS PL/I SPIE and STAE options map to Language Environment's TRAP option:

Table 12. Mapping of SPIE and STAE Options to the TRAP Option

OS PL/I	Language Environment	Action
SPIE|NOSPIE	TRAP(ON|OFF)	If SPIE|NOSPIE is specified in input, TRAP is set according to the option: TRAP(ON) for SPIE, and TRAP(OFF) for NOSPIE.
STAE|NOSTAE	TRAP(ON|OFF)	If STAE|NOSTAE is specified in input, then TRAP is set according to the option: TRAP(ON) for STAE, and TRAP(OFF) for NOSTAE.
SPIE STAE or SPIE NOSTAE or STAE NOSPIE	TRAP(ON)	If both SPIE|NOSPIE and STAE|NOSTAE are specified together in input, TRAP is set according to both options: TRAP(OFF) when both options are negative, and TRAP(ON) otherwise. TRAP(ON) must be in effect for applications to run successfully.
NOSPIE NOSTAE	TRAP(OFF)	

- The Language Environment STACK option replaces both OS PL/I ISASIZE and ISAINC options. STACK(,,ANY) can be used for the following:
  - PL/I for MVS & VM application
  - OS PL/I application relinked with Language Environment and does not contain any edited stream I/O

  Note that your application must run in AMODE(31) to use STACK(,,ANY).

  Under CICS, ALL31(ON) and STACK(,,ANY) are the defaults. However, because STACK(,,BELOW) is required for OS PL/I applications, unless it is relinked, you must change the default to STACK(,,BELOW) during installation or explicitly specify STACK(,,BELOW) for OS PL/I applications.

- The Language Environment XUFLOW option determines whether the UNDERFLOW condition is raised when underflow occurs. XUFLOW(AUTO) preserves PL/I semantics with regard to raising the UNDERFLOW condition.

The following run-time options are needed to provide compatibility with OS PL/I:

- ABTERMENC(RETCODE)
- ERRCOUNT(0)
- DEPTHCONDLMT(0)
- TRAP(ON)
- XUFLOW(AUTO | ON)

For more information about run-time options, see the *Language Environment for MVS & VM Programming Reference*.

For OS PL/I applications, the options specified in the PLIXOPT string is processed as the application-specific options. If you provide the Language Environment CEEUOPT, CEEUOPT is ignored.

Note that if the main load module contains ILC, the PLIXOPT string is ignored. In this case, you must provide CEEUOPT for the application-specific options.

Differences in Storage Report

The format, contents, and destination of the run-time storage report have changed. Language Environment provides storage information equivalent to OS PL/I. The details of storage report is described in *Language Environment for MVS & VM Programming Reference*.

The PLIXHD declaration is no longer used to provide the heading for the run-time storage report. Instead, use Language Environment's Callable Service, CEE3RPH, to specify the heading. If you do not use CEE3RPH, the heading includes the main procedure name, date, and time of execution.

Differences in Interlanguage Communication Support

There are some restrictions on support for ILC applications containing OS PL/I and other pre-Language Environment language programs. The restrictions fall into three groups:

- Fully supported load modules

  Load modules containing OS PL/I and pre-Language Environment C/370 programs are supported under Language Environment.

- Load modules you must relink

  Load modules containing OS PL/I and VS COBOL II Release 3 (or later) programs must be relinked with Language Environment.

  OS PL/I Version 2 Release 3 provides a migration aid, APARs PN69803 and PN69804, to allow you to do relinking while you are under OS PL/I Version 2 Release 3 environment. As long as the application is relinked with PN69803 and PN69804 under OS PL/I Version 2 Release 3, the application is supported under Language Environment. See “OS PL/I Object Module Relinking Tool - APARs PN69803” on page 56 for details of the migration aid.

- Unsupported ILC

  ILC between OS PL/I and the following languages is not supported:

  – Fortran
  – OS/VS COBOL
  – VS COBOL II Version 1 Release 2 or earlier releases

For more information, see *Language Environment for MVS & VM Writing Interlanguage Communication Applications*.

PL/I for MVS & VM ILC with COBOL is now reentrant. To write reentrant ILC applications, you must specify OPTIONS(REENTRANT) for all of your external procedures and ensure that you do not modify static variables. You must recompile all procedures that communicate with COBOL, using PL/I for MVS & VM.

The behavior of certain applications that use ILC might be different. For example:

- Condition handling might behave differently. The major causes of differences in condition handling are that the INTER option is now ignored, and that PL/I condition handling facilities can deal with conditions occurring in non-PL/I routines whether you specify INTER or not.
- Under OS PL/I, in applications that used ILC, the environment initialization and termination of the involved languages, including PL/I, could occur multiple times. With Language Environment for MVS & VM, there is only one run-time environment, and language-specific initialization and termination occurs only once. Changes in behavior that you might see include opening and closing of files, releasing of allocated storage, and invocation of establish ON-units.

For a complete description of how ILC works in the Language Environment for MVS & VM run-time environment, see the *Writing Interlanguage Communication Applications*.

Differences in Assembler Support

With PL/I for MVS & VM, the object module contains the CSECT name CEESTART. It also contains CEEMAIN if it has OPTIONS(MAIN) or CEEFMAIN if it has OPTIONS(FETCH). PL/I for MVS & VM no longer produces PLISTART and PLIMAIN CSECTs. CEESTART, CEEMAIN, and CEEFMAIN are not supported as a standard entry point and you cannot call them directly from an assembler program. You can call CEESTART from an assembler program only when it is a CSECT name of a PL/I for MVS & VM routine statically linked with an assembler program. Therefore, any assembler program mimicking a OS PL/I main procedure (calling PLISTART directly as a standard entry point), must continue to use PLISTART under Language Environment.

With Language Environment, assembler programs that call a PL/I routine must follow the calling conventions defined by Language Environment for MVS & VM. For example, register 13 pointing to a save area, save areas properly back-chained, and the first word of the save area being zero. For detailed information, see the *Language Environment for MVS & VM Programming Guide*.

If your OS PL/I main program is called by an assembler program and you want to convert your assembler program to use Language Environment-conforming assembler, you must either recompile your OS PL/I program without

OPTIONS(MAIN) or ensure the entry point receiving control is the real entry point of the PL/I program. In either case, the called PL/I program is treated as a subroutine. Either of these programs run under the same Language Environment enclave where the assembler program is the main program and the called PL/I program is a subroutine.

Your Language Environment-conforming assembler main program must explicitly include the Language Environment-PL/I for MVS & VM signature CSECT, CEESG010, when calling an OS PL/I subroutine to ensure the Language Environment-PL/I-specific run-time environment. There are three ways Language Environment-conforming assembler can pass control to an OS PL/I subroutine:

1. Branch to a statically linked PL/I subroutine.
2. Use the Language Environment macro CEELOAD and branch to a separately linked PL/I subroutine.
3. Use assembler instructions such as LOAD and BALR to a separately linked PL/I subroutine.

If you recompile OS PL/I subroutines that use method 1 or 2 with PL/I for MVS & VM, you don't need to include CEESG010 with your assembler program. If your assembler program uses instructions as described in method 3, you must always include CEESG010 with your assembler program, even if you recompile your PL/I subroutine with PL/I for MVS & VM.

The condition-handling behavior of the LINK from assembler is now clearly defined. For detailed information, see *Language Environment for MVS & VM Programming Guide*.

Considerations for Better CPU and Storage Utilization

After you migrate to Language Environment, you should retune your applications to maximize the performance. When you retune an application, it is not always possible to maximize CPU and storage at the same time. Often you will find that, in order to obtain better CPU, you need to use more storage, or vice versa. This section provides general tips to help you to retune your applications under Language Environment.

For more information on tools you can use to improve performance for your applications, see *Language Environment for MVS & VM Programming Guide*, *Language Environment for MVS & VM Installation and Customization under MVS*, and *PL/I for MVS & VM Programming Guide*.

Improving CPU Utilization

The following discussion shows ways to help you obtain better CPU utilization:

- Reduce the number of GETMAINs and FREEMAINs issued by Language Environment

  Use the Language Environment RPTSTG(ON) option to produce the storage report. Specify the reported storage amount in the corresponding Language Environment storage run-time options.

- Reduce the number of LOADs and DELETEs issued by Language Environment

Put the commonly used Language Environment library routines in (E)LPA. The following lists the recommended candidates for PL/I:

- CEEBINIT (LPA)
- CEEPLPKA (ELPA)
- CEEEV010 (ELPA)
- CEEBLIIA (LPA) for OS PL/I applications not relinked
- CEEOLVD (ELPA) for PL/I multitasking applications
- IBMRLIB1 (LPA)

See *Language Environment for MVS & VM Installation and Customization under MVS* for a complete list of library routines that can be put in (E)LPA.

- Avoid AMODE switching between library routines

  Use AMODE(31) for your application, if possible, so you can specify Language Environment ALL31(ON) option. If ALL31(ON) is in effect, there is no AMODE switching among library routines.

- Avoid certain PL/I conditions

  Avoid the following PL/I conditions because they might cause a slower performance to your application:

  - ENDPAGE
  - STRINGSIZE
  - AREA
  - OVERFLOW
  - ENDFILE

- Use DF/SMS-provided system-determined BLKSIZE

  On MVS, use BLKSIZE(0) for an output file that can be blocked. DF/SMS determines the optimal block size for you which can improve the file performance.

- Use Language Environment-provided math routines

  Some of the Language Environment math routines have better CPU performance than OS PL/I math routines. For most commonly used routines, Language Environment produces more accurate results than OS PL/I.

- Use Language Environment Library Routine Retention facility (LRR)

  You can get a better CPU performance if you use LRR. When LRR is used, Language Environment keeps certain Language Environment resources in storage when an application ends. Subsequent invocations of programs that use LRR is much faster because the Language Environment resources left in storage are reused.

  For example, you can use LRR for your IMS/DC environment to improve performance.

  Note that because LRR leaves LE resources in the storage for a long period of time, you must assess your storage availability to accommodate the situation.

Improving Storage Utilization

The following discussion helps you to obtain better storage utilization:

- Recompile with PL/I for MVS & VM or relink with Language Environment

  The PL/I for MVS & VM load module or the relinked OS PL/I load module has a smaller size because it contains the Language Environment stubs only.

- Make your application AMODE(31) and RMODE(ANY)

  Most likely the application will be loaded above the 16M line. You can specify the Language Environment ALL31(ON) option which allows Language Environment to allocate some of its control blocks above the 16M line.

- Use Language Environment option HEAP(,,ANY) option, if possible

  For PL/I, Language Environment will allocate the heap storage above the 16M line if the following is true:

  – The requestor is in AMODE(31)
  – HEAP(,,ANY) is in effect
  – The main program is in AMODE(31)

- Use Language Environment STACK(,,ANY) option, if possible

  Your application must be in AMODE(31). For PL/I, Language Environment will allocate the stack storage above the 16M line if one of the following is true:

  – Your application is recompiled with PL/I for MVS & VM

  – Your application is relinked with Language Environment and contains no edited stream I/O

- Reduce the IBM-supplied default values in Language Environment storage options

  If you use a smaller value, Language Environment will allocate less storage each time, but it could result in more GETMAINs and FREEMAINs being issued.

- Put commonly used Language Environment library modules in (E)LPA

  The library routines in (E)LPA do not occupy storage in your application region, so your application has more storage to use. See the recommended library routines for (E)LPA in “Improving CPU Utilization” on page 21.

Improving Performance under Subsystems

The following discussion helps you to obtain better performance under specific subsystems:

- Under CICS

  Use the PL/I FETCH/CALL statement instead of EXEC CICS LINK. The PL/I FETCH/CALL statement has a much shorter path length than the path length of EXEC CICS LINK.

- Under IMS

  Use Language Environment Library Routine Retention (LRR) facility to reduce the number of LOADs/DELETEs and GETMAINs/FREEMAINs issued by Language Environment for each transaction.

  Preload commonly used Language Environment library modules and frequently used top-level applications.

Chapter 3. Installation Considerations

This chapter contains product information you need to know at installation time. It also discusses differences in user exits, effects of Language Environment Abnormal Termination Exit, and migration of the OS PL/I Shared Library.

The Language Environment run-time options that you might want to consider at installation time are described in “Differences in Run-Time Options” on page 17.

This chapter includes the following sections:

- Product information
- Considerations for using assembler user exits
- Considerations for using high level language user exits
- Considerations for using Language Environment abnormal termination exit
- Considerations for relinking the Shared Library

Product Information

PL/I for MVS & VM has renamed its parts so that, if you want to, you can install it in the same SMP/E zone as OS PL/I. To help you identify the elements of each product, the following table lists the name differences:

Table 13. PL/I Element Names

OS PL/I	PL/I for MVS & VM
IEL0AA	IEL1AA
IKJEN00n	IEL1IKJn
IEL0nn	IEL1nn
PLInnnnn	IEL1Mnnn
PLIXnnn	IEL1nnn
PLIHELP	IEL1PLIH

Language Environment must be available before you can compile, link-edit, and run a PL/I for MVS & VM application. If you attempt to compile a program before installing Language Environment, the program will not compile and a message will be generated. A STEPLIB concatenation for SCEERUN must be added in the compile. The details of the datasets and modules shipped with PL/I and Language Environment can be found in one of the documents listed below. If you want to know the datasets and modules names, storage requirements, or other details specifically for installation planning, refer to one of these documents:

PL/I for MVS & VM Installation and Customization under MVS
PL/I for MVS & VM Program Directory under VM
Language Environment for MVS & VM Installation and Customization under MVS
Language Environment for MVS & VM Program Directory under VM

There are additional requirements you need to be aware of before you begin to use PL/I for MVS & VM and Language Environment. These requirements are different for each system and are described in the following sections.

If you plan to use the Debug Tool with your PL/I applications, you must have Language Environment for MVS & VM Release 4 installed on your system.

MVS Requirements

You must have access to Language Environment when you compile your PL/I for MVS & VM application. When you compile your application under MVS and you use existing JCL, be sure your STEPLIB or JOBLIB statement includes SCEERUN (Language Environment run-time library), unless you are using TASKLIB or LINKLIB which already includes SCEERUN. You can use the IEL1C cataloged procedure to compile PL/I applications, or you can continue to invoke the PLIOPT module directly.

Your compile step should include the following:

```
//PLI      EXEC PGM=IEL1AA,PARM='OBJECT,NODECK',REGION=512K
//STEPLIB  DD  DSN=IEL.V1R1M1.SIELCOMP,DISP=SHR
//         DD  DSN=CEE.V1R4M0.SCEERUN,DISP=SHR
```

Reading about the cataloged procedures provided with PL/I for MVS & VM can help you understand the use of SCEERUN during compilation. “Using PL/I Cataloged Procedures under MVS” is a chapter in *PL/I for MVS & VM Programming Guide*.

When you link-edit your PL/I for MVS & VM application or relink your OS PL/I application with Language Environment and you use existing JCL, be sure your SYSLIB statement includes SCEELKED (Language Environment link-time library). Language Environment also provides the following three libraries to which you can link for specific PL/I functions and compatibility support:

SIBMMATH Provides the same results as if you were using the OS PL/I Version 2 Release 3 math library.

SIBMCALL Provides the support for OS PL/I PLICALLA and PLICALLB entry points.

SIBMTASK Provides the support for PL/I multitasking.

You can use any or all of these libraries at the same time and they can appear in any order, as long as they are specified before SCEELKED.

You must specify SYSLIB if you plan to use it. Do not include SYSLIB unless you are using a TASKLIB or LINKLIB which already includes SCEELKED.

If you plan to run your multitasking applications, you must have MVS/ESA SP V5R1 (or later) with OpenEdition MVS Services installed on your system.

VM Requirements

When you compile your application, you must link to the minidisk that contains Language Environment for MVS & VM run-time loadlib SCEERUN. You must include the SCEERUN loadlib in a GLOBAL LOADLIB command before you invoke the compiler. You can use the IEL1PLI EXEC to compile PL/I applications, or you can continue to invoke the PLIOPT module directly.

When you link-edit your PL/I for MVS & VM application or relink your OS PL/I application, you must link to the minidisk that contains the Language Environment for MVS & VM link-time txtlib SCEELKED. You must include SCEELKED in the GLOBAL TXTLIB when you load your application. Language Environment also

provides the following three libraries to which you can link for specific PL/I functions and compatibility support:

SIBMMATH Provides the same results as if you were using the OS PL/I Version 2 Release 3 math library.

SIBMCALL Provides the support for OS PL/I PLICALLA and PLICALLB entry points.

SIBMTASK Provides the support for PL/I multitasking.

You can use any or all of these libraries at the same time and they can appear in any order, as long as they are specified before SCEELKED.

When you run your PL/I for MVS & VM or OS PL/I application,. you must link to the minidisk that contains the Language Environment run-time loadlib SCEERUN. You must include SCEERUN in the GLOBAL LOADLIB when you run your application.

Considerations for Using Assembler User Exits

The OS PL/I Version 2 assembler user exits IBMBXITA and IBMFXITA are supported for compatibility reasons only. Use the Language Environment user exit CEEBXITA as a replacement.

Compatibility Considerations

This section describes the restrictions on support for IBMBXITA, IBMFXITA, and CEEBXITA in different types of load modules:

Relinked OS PL/I Version 2 MAIN load module

Under MVS and VM, when the main procedure is relinked with Language Environment for MVS & VM, you can use either an Language Environment for MVS & VM-defined, application-specific CEEBXITA user exit or the existing IBMBXITA user exit. If you use IBMBXITA, you must use the linkage editor INCLUDE statement to explicitly include IBMBXITA in the MAIN load module. You can include both CEEBXITA and IBMBXITA in the MAIN load module. If Language Environment for MVS & VM finds both CEEBXITA and IBMBXITA at run time, CEEBXITA is given control.

Under CICS, when the main procedure is relinked with Language Environment for MVS & VM, only CEEBXITA is supported. The application-specific CEEBXITA must be linked with the MAIN load module; it affects the newly created enclave only.

OS PL/I Version 2 MAIN load module

Under MVS and VM, only IBMBXITA is supported. The load module always contains a copy of IBMBXITA—either an application-specific one or the default one provided by OS PL/I Version 2.

Under CICS, only IBMFXITA is supported. IBMFXITA can be provided at installation time only. You must use the linkage editor INCLUDE statement to explicitly include the desired IBMFXITA into IBMRSAP at installation time. In this case, IBMFXITA affects every OS PL/I Version 2 transaction in the CICS region. If IBMFXITA is not included in IBMRSAP, the Language Environment for MVS & VM-defined default CEEBXITA is given control for every OS PL/I Version 2 transaction in the CICS region.

PL/I for MVS & VM MAIN load module
Under MVS, VM, and CICS, only CEEBXITA is supported.

For OS PL/I main load module or relinked OS PL/I main load module, the following rules of precedence show in Table 14 is used to determine which assembler exit is given control during run time.

Table 14. Rules of Precedence for Assembler User Exits at Run Time

CEEBXITA present	IBMBXITA present under MVS or VM; IBMFXITA present under CICS	Exit driven
No	No	Default version of CEEBXITA
Yes	No	CEEBXITA
No	Yes	IBMBXITA under MVS and VM; IBMFXITA under CICS
Yes	Yes	CEEBXITA

Changes to Assembler User Exits

The values in some parameters of IBMBXITA and IBMFXITA are supported in the same way as in previous releases. The following list describes the parameters.

CXIT_LEN
The value of CXIT_LEN is 48.

CXIT_FUNC
Support for CXIT_FUNC has changed as follows:

- The value for both initial and nested enclave initialization is 1.
- The value for both initial and nested enclave termination is 2.
- Neither IBMBXITA nor IBMFXITA is called for process termination.

CXIT_RETURN
On entry, CXIT_RETURN contains the Language Environment-defined enclave return code. This enclave return code is the PL/I return code plus the Language Environment return code modifier.

The value might be different from the value that would have been passed under OS PL/I Version 2. For VM, the Language Environment return code modifier contains six digits. Further, the severity of some conditions is different between OS PL/I and Language Environment. On return, the function of CXIT_RETURN remains unchanged.

When you migrate your applications from OS PL/I Version 2, remember that the CXIT_RETURN value and the CXIT_REASON value can be different from those in OS PL/I Version 2. If IBMBXITA or IBMFXITA is dependent on a particular nonzero CXIT_RETURN or CXIT_REASON value, you must modify that dependency in IBMBXITA or IBMFXITA to match the value in Language Environment.

CXIT_REASON
Values defined by OS PL/I Version 2 for CXIT_REASON are no longer supported. On entry, CXIT_REASON contains the Language Environment-defined return code modifier. The value can be different from the value that would have been passed under OS PL/I Version 2. See CXIT_RETURN on page 27 for explanation of dependency.

On return, the function of CXIT_REASON remains unchanged.

CXIT_FLAGS
The definitions for CXIT_ABTRM, CXIT_ABND, and CXIT_DUMP are the same as they were in OS PL/I Version 2. The entire OS task terminates abnormally under MVS if you set CXIT_ABND to 1.

For more information, see *OS PL/I Version 2 Release 3 Programming Guide*.

CXIT_PARM
The CXIT_PARM parameter value has changed as follows:

- On entry, CXIT_PARM contains an address pointing to your user parameter list, without run-time options, from the invocation level of the main program. The user parameter list has already been processed based on parameter list style rules.
- On return, the value in CXIT_PARM is saved as a user parameter and later passed to the main program.

CXIT_WORK
CXIT_WORK is unchanged.

CXIT_OPTIONS
Under MVS and VM, CXIT_OPTIONS can contain an options list that you provide. The rules of precedence for merging the options list are the same as they were in OS PL/I Version 2, but the options are supported according to the rules defined in Language Environment (see *Language Environment for MVS & VM Programming Guide*). Under CICS, CXIT_OPTIONS remains unsupported.

CXIT_USERWD
CXIT_USERWD retains its OS PL/I Version 2 function, except that the USERWD is passed without alteration to every user exit across multiple enclaves.

CXIT_CODES
CXIT_CODES retains its OS PL/I Version 2 function; that is, it specifies a list of abend codes that are handled by the Language Environment condition handler. You must include the abends 777, 778, and U3501 in CXIT_CODES so your database can roll back under IMS or DB2.

CXIT_PAGE
The IBM-supplied default is still 32K. CXIT_PAGE is unchanged.

When you migrate your applications from OS PL/I Version 2 Release 3, remember that the displacement of CXIT_PAGE in IBMBXITA and IBMFXITA under OS PL/I Version 2 Release 3 is different from that of CEEBXITA. You cannot rename these user exits to CEEBXITA as a shortcut.

Specific Considerations

- Under VM, the IBM-supplied default CEEBXITA issues FILEDEFs for SYSIN, SYSOUT, and CEEDUMP without the PERM option. The default CEEBXITA no longer issues FILEDEFs for SYSPRINT and PLIDUMP. However, if you do not supply your own FILEDEF for SYSPRINT, the VM default file is used during OPEN. If the dump file is not present during OPEN, Language Environment dynamically allocates the CEEDUMP file.
- If you use the MSGFILE(SYSPRINT) option so run-time messages are directed to the STREAM PRINT OUTPUT file, and your IBMBXITA clears the SYSPRINT file during termination, you must remove the FILEDEF CLEAR statement of the SYSPRINT file from IBMBXITA. Otherwise, the SYSPRINT

file is treated as a process resource and will not be cleared until Language Environment process termination. If IBMBXITA is in effect, it is given control for Language Environment enclave initialization and termination only.

- The PLIDUMP or CEEDUMP file for the dump output is also treated as a process resource and must not be cleared during enclave termination. If your IBMBXITA contains the FILEDEF CLEAR for the dump file during termination, you must remove the FILEDEF CLEAR for the dump file. If you do not remove the FILEDEF CLEAR for the dump file, the result is unpredictable.
- The OS PL/I abend exit IBMBEER is ignored under Language Environment. See "Differences in Condition Handling" on page 13 for forcing an abend under Language Environment.

For more information on assembler language user exits, see the *Language Environment for MVS & VM Programming Guide*.

Considerations for Using High-Level Language User Exits

The OS PL/I Version 2 High-Level Language (HLL) user exit IBMBINT is no longer recommended; it is only supported for compatibility. Instead, you should use the Language Environment HLL user exit CEEBINT.

The way in which IBMBINT is given control now depends on the version of PL/I in which it is used. The following describes support for IBMBINT:

Relinked OS PL/I Version 2 Release 2 or Release 3 MAIN load module
Only CEEBINT is supported. To continue to use IBMBINT, you must rename IBMBINT to CEEBINT. CXIT_USERWD is shared across multiple enclaves.

OS PL/I Version 2 Release 2 or Release 3 MAIN load module
Only IBMBINT is supported. The load module always contains a copy of IBMBINT, either the application-specific one or the default one provided by OS PL/I Version 2 Release 2 or Release 3.

PL/I for MVS & VM MAIN load module
Only CEEBINT is supported. The load module always contains a copy of CEEBINT, either the application-specific one or the default one provided by Language Environment.

If you write CEEBINT in PL/I, you must write it in PL/I for MVS & VM, OS PL/I Version 2 Release 2 or Release 3. If CEEBINT calls any PL/I routines, those routines must also be written in one of these language releases.

Do not use the OPTIONS(MAIN) statement in the PL/I HLL user exit.

The STOP statement terminates the application.

Considerations for Using Language Environment Abnormal Termination Exit

Language Environment provides an abnormal termination exit for an application terminating with an unhandled condition of severity 2 or greater. The exit allows you to collect problem determination data before Language Environment frees the resources it has acquired. This exit is available under MVS only.

With the default abnormal termination exit, CEEBDATX, you will receive the U4039 abend and optionally a system dump if you provides a SYSABEND or SYSUDUMP DD card. The application then continues its termination with a return code if ABTERMENC(RETCODE) is in effect or an abend code if ABTERMENC(ABEND) is in effect.

If you do not want the abnormal termination exit to get control, or you want the abnormal termination exit to do something else such as BR 14, when you install Language Environment, you must update the CEEEXTAN CSECT in SCEERUN by changing the CEEXART macro and then run the CEEWDEXT (for CICS it's CEEWCEXT) JCL. See *Language Environment for MVS & VM Installation and Customization under MVS* for details.

Language Environment sample library SCEESAMP contains a sample exit, CEEBNATX, which simply does BR 14. You can assemble, name CEEBNATX to CEEBDATX, and put this CEEBDATX into a dataset that is concatenated before SCEERUN. In this case, SCEERUN still contains the default CEEBDATX.

Considerations for relinking the Shared Library

The OS PL/I Shared Library must be replaced with Language Environment stubs in order to provide support for OS PL/I Version 1 Release 5.1 and Version 2 load modules that use the Shared Library. IBM has provided you with sample jobs (IBMRLSLB for OS PL/I Version 1 Release 5.1 non-CICS nonmultitasking Shared Library and IBMRLSLA for the other releases or multitasking Shared Library) located in SCEESAMP that help you replace your Shared Library with Language Environment stubs.

OS PL/I applications that use the Shared Library are supported by Language Environment if the following conditions are true:

- The Shared Library is OS PL/I Version 1 Release 5.1 or Version 2
- The Shared Library was created with all PLRSHR options
- The Shared Library is replaced with Language Environment stubs

If any of the above conditions is not true, the application is not supported. You must relink your application with Language Environment or OS PL/I Version 2. Once the application is relinked, it no longer uses the OS PL/I Shared Library feature.

Note that you must also observe the rules of supporting OS PL/I object and load modules under Language Environment. See Chapter 4, “Object and Load Module Considerations” on page 32 for details.

When you replace your Shared Library with Language Environment stubs using the sample JCL IBMRLSLA or IBMRLSLB, you need to consider the following concatenations:

- SIBMMATH before SCEELKED if OS PL/I math routine is desired
- SIBMTASK before SCEELKED if multitasking Shared Library is used

Once the Shared Library is replaced with Language Environment stubs, you cannot apply OS PL/I maintenance to it. The AMODE and RMODE of the replaced Shared Library must remain the same unless all applications that use the Shared Library have changed to a different set of AMODE and RMODE.

See “OS PL/I Shared Library Replacement Tool” on page 55 for details of how to use IBMRLSLA and IBMRLSLB.

Chapter 4. Object and Load Module Considerations

This chapter describes factors that affect the compatibility of OS PL/I object and load modules in the Language Environment for MVS & VM environment. It discusses the following types of considerations:

- OS PL/I Version 1 object and load module compatibility
- OS PL/I Version 2 object and load module compatibility
- Summary of support for OS PL/I

All of the library routines in a load module must be from the same release of the run-time library. For example, Language Environment stubs, OS PL/I Shared Library stubs, and OS PL/I resident library routines cannot exist in the same load module.

OS PL/I Version 1 Object Module and Load Module Compatibility

Language Environment supports object modules and load modules for OS PL/I Version 1 with some restrictions. You can continue to use most of your Version 1 object and load modules if you observe the rules described in the following sections.

If a load module contains an OS PL/I Version 1 object module but is linked with OS PL/I Version 2 resident library, the load module is considered an OS PL/I Version 2 load module and the rules for OS PL/I Version 2 apply. However, if the load module contains OS PL/I Version 1 Release 1.0 - 2.3 object modules, the object module must be recompiled.

If a load module contains the OS PL/I abend exit, IBMBEER, the abend exit is ignored by Language Environment. See “Considerations for Using Assembler User Exits” on page 26 for more information on this topic.

OS PL/I Version 1 Release 5.1

Object Module

The object module is supported.

Load Module Not Using Shared Library:

- Main load module for MVS non-CICS nonmultitasking

  The OS PL/I bootstrap routine, IBMBPIRA, always linked with a user load module, contains features such as the fast initialization and termination that are not compatible with Language Environment. A sample ZAP, IBMRZAPM, is provided in Language Environment SCEESAMP to help you deactivate those incompatible features. The sample ZAP is described in “OS PL/I Version 1 Release 5.1 Main Load Module ZAP” on page 54.

  ZAPped load modules continue to work under OS PL/I V1.5.1 and V2, as well as Language Environment. However, performance degradation might occur if the original load module contains the fast initialization and termination feature.

  If you do not ZAP your load module, you must do one of the following:

  - Relink your object module with Language Environment or OS PL/I Version 2
  - Use the OS PL/I Library Routine Replacement Tool to replace the library routines in the load module with Language Environment stubs

- Main load module for MVS non-CICS multitasking

  The load module is supported.

- Main load module under CICS

  The load module is supported.

- Main load module under VM

  The OS PL/I VM-specific bootstrap routine, DMSIBM, contains features that are not compatible with Language Environment. A sample ZAP, IBMRZAPV, is provided in Language Environment SCEESAMP to help you deactivate the incompatible features. The sample ZAP is described in "OS PL/I Version 1 Release 5.1 Main Load Module ZAP" on page 54.

  The ZAPped load module is supported under Language Environment only. It **no longer** works under OS PL/I Version 1 or Version 2. If you do not ZAP your load module, you must do one of the following:

  - Relink your object module with Language Environment or OS PL/I Version 2
  - Use the OS PL/I Library Routine Replacement Tool to replace the library routines in the load module with Language Environment stubs

    See "OS PL/I Library Routine Replacement Tool" on page 53 for a description of this tool.

- FETCHed subroutine load module

  The load module is supported.

Load Module Using the Shared Library

The load module is supported as long as the OS PL/I V1R5.1 Shared Library was created with all PLRSHR options and the Shared Library, including the multitasking Shared Library, is replaced with Language Environment stubs. The Shared Library needs to be replaced only once during Language Environment installation.

If the Shared Library was not created with all PLRSHR options or the Shared Library is not replaced with Language Environment stubs, the object module must be relinked with Language Environment or OS PL/I Version 2, or you can replace the Shared Library stubs in the load module with Language Environment stubs. After the object module is relinked or the load module is replaced, the OS PL/I Shared Library feature is no longer used.

OS PL/I Version 1 Release 5.0

OS PL/I Version 1 Release 5.0 provides support only for MVS applications. VM and CICS are not supported in Release 5.0.

Object Module

The object module is supported.

Load Module

The load module is not supported, whether or not you use the Shared Library. You must relink your object module with Language Environment or OS PL/I Version 2, or you can use the OS PL/I Library Routine Replacement Tool to replace the library routines in the load module with Language Environment stubs. See "OS PL/I Library Routine Replacement Tool" on page 53 for a description of this tool.

OS PL/I Version 1 Release 3.0 - Release 4.0

Object Module

- Under MVS

  The object module is supported except for the CICS macro language.

- Under VM

  The object module is supported.

Load Module

The load module is not supported, whether or not you use the Shared Library. You must relink your object module with Language Environment or OS PL/I Version 2, or you can use the OS PL/I Library Routine Replacement Tool to replace the library routines in the load module with Language Environment stubs. See "OS PL/I Library Routine Replacement Tool" on page 53 for a description of this tool.

OS PL/I Version 1 Prior to Release 3.0

Object modules or load modules created prior to Release 3.0 are not supported and you must recompile your application with PL/I for MVS & VM or OS PL/I Version 2.

OS PL/I Version 2 Object Module and Load Module Compatibility

Object modules and load modules created with OS PL/I Version 2 can run without relinking if they do not contain any features that are not supported or that require relinking.

Language Environment supports OS PL/I applications that contain the PL/I assembler user exit, IBMxXITA. See "Considerations for Using Assembler User Exits" on page 26 for more information on this topic.

Summary of Support for OS PL/I Object and Load Modules

The following table summarizes the PL/I object- and load-module support described in this chapter:

Table 15. Summary of Object and Load Module Support by Language Environment

Support description	V2	V1R5.1	V1R5.0	V1R3.0-V1R4.0	Prior to V1R3.0
Main load module	Yes[3]	Yes[1,3]	No	No	No
Fetched subroutine load module	Yes[3]	Yes[3]	No	No	No
Object module	Yes	Yes	Yes	Yes[2]	No

[1]Use OS PL/I Version 1.5.1 load module ZAP for MVS non-CICS nonmultitasking or VM load modules
[2]CICS macro language is not supported.
[3]Shared Library must be created with all PLRSHR options and must be replaced with Language Environment stubs.

Chapter 5. Link-Edit Considerations

This chapter describes factors you must consider when you link-edit an object module produced by different releases of OS PL/I and PL/I for MVS & VM. It includes the following sections:

- Symbol table considerations
- NCAL linkage editor option
- GENMOD for VM
- Using OS PL/I math routines
- Using multitasking
- Using OS PL/I PLICALLA or PLICALLB entry

Symbol Table Considerations

If you link-edit an object module produced by different releases of PL/I, and the object module contains symbol tables for external variables, the symbol table that appears in the resultant load module must be the one produced by the most recent release of PL/I.

The compiler produces an object module that contains external symbol table control sections (CSECTs) if your program includes one or more of the following PL/I features:

- GET DATA statements for external variables
- PUT DATA statements for external variables
- The TEST(SYM) compile-time option for external variables

If your program uses one or more of these features with external variables, you must ensure that the correct symbol table appears in your load module. Place the object module produced by the most recent release of PL/I ahead of all other object modules in the link-edit job stream. If more than one object module produces a symbol table CSECT with the same name, the linkage editor keeps the symbol table CSECT that it encounters first and discards the other symbol tables.

For example, suppose you link-edit an object module produced by OS PL/I Version 1 Release 5.1 with an object module produced by PL/I for MVS & VM. Put the object module produced by PL/I for MVS & VM ahead of the object module produced by OS PL/I Version 1 Release 5.1 in the link-edit job stream. By doing this, the linkage editor keeps the symbol table produced by PL/I for MVS & VM if both object modules produce symbol tables.

NCAL Linkage Editor Option

Under Language Environment, the NCAL linkage editor option is no longer required when you link-edit your subroutine object modules for the future use. This is true for both PL/I for MVS & VM and OS PL/I produced object modules. Once these object modules are linked with Language Environment, they contain Language Environment stubs. Those stubs are compatible among Language Environment releases.

Note that a load module must not contain Language Environment stubs and OS PL/I resident library routines.

GENMOD for VM

If you are using GENMOD to create VM modules, use the RLDSAVE option in the LOAD and INCLUDE commands. The PL/I for MVS & VM object module nominates the CEESTART CSECT as the main entry point. However, if you are using GENMOD with a combination of OS PL/I and PL/I for MVS & VM object modules, specify FROM PLISTART if the main program is OS PL/I or FROM CEESTART if the main program is PL/I for MVS & VM.

PL/I programs that use the VM LOAD and INCLUDE commands can specify the HOBSET, HOBSETSD, or NOHOBSET options. Before you use HOBSET or HOBSETSD, understand the special considerations and restrictions that apply to these options. For detailed information, see the *Language Environment for MVS & VM Programming Guide*. There are no special considerations or restrictions for NOHOBSET.

Using OS PL/I Math Routines

Language Environment provides a set of math routines, including routines for exponentiation. For most commonly used routines, Language Environment produces more accurate results than OS PL/I. Some of Language Environment routines also have better performance than OS PL/I. You should use Language Environment-provided math routines.

Language Environment also provides the OS PL/I math routines to help you to migrate to Language Environment. However, the OS PL/I math routines are provided for compatibility only and will be withdrawn in the future.

If your application must use the OS PL/I math routines under Language Environment, when you link-edit your object module (produced by OS PL/I or PL/I for MVS & VM), place SIBMMATH in front of SCEELKED.

When you run your PL/I for MVS & VM or OS PL/I application under Language Environment and you use existing JCL, be sure your STEPLIB or JOBLIB statement includes SCEERUN, unless you TASKLIB or LINKLIB which already includes SCEERUN.

Using Multitasking

When you link-edit a multitasking object module, produced by OS PL/I or PL/I for MVS & VM, with Language Environment, you must concatenate SIBMTASK in front of SCEELKED. See “Differences in Multitasking Support” on page 10 for details.

Using OS PL/I PLICALLA or PLICALLB Entry

If you recompile every PL/I program with PL/I for MVS & VM in the main load module that uses OS PL/I PLICALLA or PLICALLB as the main entry point, when you link-edit the object modules, you might need to place SIBMCALL in front of SCEELKED. See “PLICALLA Considerations” on page 4 and “PLICALLB Considerations” on page 6 for details.

Chapter 6. Compile-Time Considerations

This chapter describes the following compile-time considerations:

- Dependency on Language Environment for MVS & VM
- Large arrays and aggregates
- Compatibility considerations for OS PL/I Version 1 source code
- Differences in user return codes
- Storage report changes
- Compiler message changes
- Messages that PL/I issues for errors in the PLIXOPT string

Dependency on Language Environment for MVS & VM

Language Environment for MVS & VM must be available whenever you compile a PL/I application.

Large Arrays and Aggregates

If all of your existing object modules were produced by OS PL/I Version 2 with the CMPAT(V2) compile-time option, your object module is fully compatible with object modules produced by PL/I for MVS & VM, provided you continue to use the CMPAT(V2) compile-time option.

If some or all of your existing object modules were produced by OS PL/I Version 2 with the CMPAT(V1) compile-time option or by OS PL/I Version 1 Release 5.1, the following considerations apply when you link-edit:

- If arrays, aggregates, or AREAs are to be shared between OS PL/I Version 1 Release 5.1 or OS PL/I Version 2 (compiled with the CMPAT(V1) option) object modules and PL/I for MVS & VM object modules, PL/I for MVS & VM compilations must use the CMPAT(V1) option.
- If arrays, aggregates, or AREAs are to be shared between OS PL/I Version 2 (compiled with the CMPAT(V2) option) object modules and PL/I for MVS & VM object modules, PL/I for MVS & VM compilations must use the CMPAT(V2) option.

You must use the CMPAT(V2) compile-time option to use arrays, aggregates or AREAs larger then 32K. The CMPAT(V2) option is recommended even if you do not use larger arrays, aggregates, or AREAs.

Figure 1 on page 39 shows the differences between compiling with CMPAT(V1) and CMPAT(V2). For more information about the CMPAT option, see the *PL/I for MVS & VM Programming Guide*.

```
/* Built-in functions DIM, HBOUND, LBOUND and ALLOCATION return     */
/* fullword values when compiled with CMPAT(V2) and return halfword */
/* values when compiled with CMPAT(V1).                             */


DC510: PROC OPTIONS(MAIN);

DCL BA219(25:28) BIT(4) AUTOMATIC VARYING ALIGNED;
DCL BU114(2,2) BIT(80) CONTROLLED UNALIGNED;

ALLOCATE BU114;

CALL INT;

INT: PROC;
  DCL BA221(1092:1095) BIT(15) ALIGNED INIT(HBOUND(BA219,1),
                                        LBOUND(BA219,1),
                                      ALLOCATION(BU114));
BA221(1095) = ALLOCATION(BU114);
PUT DATA (BA221(1092),BA221(1093),BA221(1094),BA221(1095));

DCL BA222(1092:1095) BIT(31) ALIGNED INIT(HBOUND(BA219,1),
                                        LBOUND(BA219,1),
                                     ALLOCATION(BU114));
BA222(1095) = ALLOCATION(BU114);
PUT DATA (BA222(1092),BA222(1093),BA222(1094),BA222(1095));
END INT;

END DC510;


Run-time output if compiled with CMPAT(V1):

 BA221(1092)='000000000011100'B                  BA221(1093)='000000000011001'B
 BA221(1094)='000000000000001'B                  BA221(1095)='000000000000001'B;
 BA222(1092)='00000000001110000000000000000000'B  BA222(1093)='00000000001100100000000000000000'B
 BA222(1094)='00000000000000100000000000000000'B  BA222(1095)='00000000000000100000000000000000'B;


Run-time output if compiled with CMPAT(V2):

 BA221(1092)='000000000000000'B                  BA221(1093)='000000000000000'B
 BA221(1094)='000000000000000'B                  BA221(1095)='000000000000000'B;
 BA222(1092)='00000000000000000000000000011100'B  BA222(1093)='00000000000000000000000000011001'B
 BA222(1094)='00000000000000000000000000000001'B  BA222(1095)='00000000000000000000000000000001'B;
```

Figure 1. Differences between Compiling with CMPAT(V1) and CMPAT(V2)

Compatibility Considerations for OS PL/I Version 1 Source Code

Source code compatibility with Version 1 is supported with the following exceptions:

- CHARSET(48) and CHARSET(BCD) are no longer supported.
- Graphic DBCS varies slightly from old EGCS in that the shift-in and shift-out code points are fixed.
- When using CMPAT(V2), the following items cause incompatibilities at run time:
  - Arrays (including arrays of structures and structures of arrays), whether these have fullword subscripts or not.
  - BASED and CONTROLLED AREAs and aggregates, whether these are larger than 16 megabytes or not.

  - Expressions that use values returned by the HBOUND, LBOUND, DIM, and ALLOCATION built-in functions. These built-in functions now return FIXED BIN(31) results instead of FIXED BIN(15) results. However, if you specify CMPAT(V1), they return FIXED BIN(15) results.

- Processing of %INCLUDE statements now delimits text inclusions with “begin” and “end” comments.
- The preprocessor now treats character codes outside the range of '40'X through 'FF'X as delimiters if they are not part of a string constant.
- Suffixes that follow string constants are not replaced by the preprocessor—whether these are legal PL/I suffixes or not—unless you insert a delimiter between the ending quotation mark of the string and the first letter of the suffix. For example:

```
%DCL (GX, XX) CHAR;
%GX='||FX';
%XX='||ZZ';
DATA = 'STRING'GX;
DATA = 'STRING'XX;
DATA = 'STRING' GX;
DATA = 'STRING' XX;
```

  under Version 1 produces the source:

```
DATA = 'STRING'||FX;
DATA = 'STRING'||ZZ;
DATA = 'STRING' ||FX;
DATA = 'STRING' ||ZZ;
```

  whereas, under Version 2 it produces:

```
DATA = 'STRING'GX;
DATA = 'STRING'XX;
DATA = 'STRING' ||FX;
DATA = 'STRING' ||ZZ;
```

Differences in User Return Codes

The PL/I built-in functions that support user return codes now have a precision of FIXED BIN(31) under PL/I for MVS & VM. This change affects the following:

- You can use the PLIRETC built-in subroutine to set return codes with a value greater than 999 under PL/I for MVS & VM.

  In OS PL/I, PLIRETC does not accept values greater than 999. If you set the value greater than 999, PLIRETC resets the value to 999 and issues an informational message. You must recompile your program with PL/I for MVS & VM to take advantage of the fullword return code.
- The OS PL/I PLIRETV built-in function returns a FIXED BIN(15) value. You must relink with Language Environment to take advantage of the fullword return code.
- The OS PL/I RETCODE option saves the return code in the lower two bytes of register 15. PL/I for MVS & VM uses a fullword in register 15 for the return code. If you use the RETCODE option and your application is a mixture of PL/I for MVS & VM and OS PL/I Version 2 Release 3, Language Environment returns a halfword return code when the option is found in OS PL/I and a fullword return code when the option is found in PL/I for MVS & VM, even

though you relink your application. You must recompile your program with PL/I for MVS & VM to receive a fullword return code when using the RETCODE option.

User return codes are also discussed in “Differences in User Return Code” on page 13.

Storage Report Changes

The PLIXHD variable is no longer used as the heading in storage reports. The identifier PLIXHD is no longer special; you can declare it and use it as you would declare and use any other variable.

Compiler Message Changes

This section lists the numbers of PL/I compiler messages that have changed. For detailed descriptions of messages, see *PL/I for MVS & VM Compile-Time Messages and Codes*.

Notes:

1. The messages produced for run-time options specified in the PLIXOPT string are different. In most cases, these messages now refer you to the description of a similar Language Environment for MVS & VM run-time message. For more information about messages produced for run-time options specified in the PLIXOPT string, see “Messages That PL/I Issues for Errors in the PLIXOPT String” on page 42.
2. The severity of the preprocessor messages IEL0115I, IEL0163I, and IEL0201I has been reduced to W.
3. The compiler no longer issues message IEL0983I for Language Environment callable service names.

The following compiler messages have been added:

IEL0040I	IEL0573I	IEL0952I
IEL0048I	IEL0574I	IEL0953I
IEL0537I	IEL0575I	IEL0985I
IEL0558I	IEL0576I	IEL0995I
IEL0566I	IEL0577I	IKJ65080I
IEL0567I	IEL0756I	IKJ65081I
IEL0570I	IEL0936I	IKJ65082I
IEL0571I	IEL0950I	IKJ65083I
IEL0572I	IEL0951I	

The following compiler messages have been changed:

IEL0001I	IEL0233I	IEL0966I
IEL0115I	IEL0670I	IEL0967I
IEL0163I	IEL0772I	IEL0970I
IEL0201I	IEL0809I	IKJ65035I
IEL0230I	IEL0929I	IKJ65059I

The following compiler messages are no longer valid:

IEL0043I	IEL0726I	IEL0959I
IEL0430I	IEL0954I	IEL0983I
IEL0547I	IEL0958I	

Messages That PL/I Issues for Errors in the PLIXOPT String

The PLIXOPT variable is a varying-length character string that contains run-time options you can specify at compile time. The messages that the compiler produces to diagnose errors in these options have changed. In most cases, the PL/I messages now list an associated Language Environment message that you should read for more information about the error.

PL/I parses the PLIXOPT string and produces the Language Environment CEEUOPT csect. If you explicitly include CEEUOPT in your recompiled application ahead of the compiler-generated CEEUOPT CSECT, the explicitly included CEEUOPT CSECT will override the one generated by the compiler for the options specified in the PLIXOPT string.

The following messages describe different types of problems that can occur with the run-time options specified in a PLIXOPT string.

Message IEL0950I (Warning)
A severe error occurred in the PLIXOPT string. Generally, this message indicates that the error is severe enough to cause the parsing of the string to fail.

You must correct the errors that cause this message.

Message IEL0951I (Warning)
An error occurred in a run-time option in the PLIXOPT string. This message indicates that the string contains an item that is not a valid run-time option. This message is issued for items that are not recognized as valid run-time options, including run-time options that are no longer supported.

You must correct the errors that cause this message.

Message IEL0952I (Informational)
A possible problem exists with a run-time option in the PLIXOPT string. This message is issued for OS PL/I run-time options that have been replaced with similar Language Environment run-time options. The OS PL/I options are automatically converted to the appropriate Language Environment options, but some of the Language Environment options might not function exactly as the OS PL/I options did.

You should convert these OS PL/I options to the appropriate Language Environment option, and check to see if the Language Environment for MVS & VM option is different from the OS PL/I option.

Message IEL0953I (Informational)
A possible incompatibility exists in the support for a run-time option in the PLIXOPT string. This message is issued for OS PL/I run-time options that have been replaced by similar Language Environment run-time options, but might not be supported exactly as OS PL/I supported them.

For example, this message is issued for the OS PL/I NOSPIE and NOSTAE options because both options map to the Language Environment TRAP option. TRAP(ON) implies both SPIE and STAE, and TRAP(OFF) implies both NOSPIE and NOSTAE; under Language Environment, there is no support that is equivalent to the support that OS PL/I provided for the combinations SPIE and NOSTAE, or NOSPIE and STAE. To see how SPIE and STAE map to TRAP, see Table 12 on page 18.

Chapter 7. Subsystem Considerations

This chapter discusses subsystem-specific considerations that you need to know when you migrate your applications running under CICS, IMS, and DB2.

CICS Considerations

Language Environment provides the same level of OS PL/I object and load module support as for non-CICS. See Chapter 4, “Object and Load Module Considerations” on page 32 for details. If you are running under CICS Version 3 Release 3, you must ensure the CICS APAR PN38032 is installed. Without PN38032, your application trying to use Language Environment will receive the APLE abend.

The CICS Storage Protect facility was introduced under CICS 3.3. This provides more data integrity and security for the application program and especially for the entire CICS region. Because of the new feature, you might discover that some of the successfully running OS PL/I applications start to fail with ASRA(0C4) abend and the CICS message DFHSR0622.

If the above problem is happening in your OS PL/I application program, either of the following two methods might be able to fix your problem:

1. Set the CICS system initialization parameter RENTPGM=NOPROTECT. This sets the protection of the user program in user key. Notice the default for RENTPGM is PROTECT.
2. Relink your OS PL/I application program under Language Environment with APAR PN38032 installed.

If the stream output function is used in your OS PL/I CICS application, especially the `PUT DATA;` statement, it might trigger the above error. PL/I stream output function is intended for debugging purposes only. For the performance reason, it is recommended not to be included in the production programs.

Updating CICS System Definition (CSD) File

When you bring up a CICS region with Language Environment, you must ensure the module names listed in Language Environment CEECCSD are defined in the CSD. You can locate CEECCSD in SCEESAMP. If you use CICS Version 4 autoinstall facility, you do not need to define Language Environment modules manually in the CSD.

Error Handling

A diagnostic message is issued only if there is no ERROR ON-unit established in the program, or the ERROR ON-unit does not recover from the condition by using a GOTO out of block.

Support for IBMFXITA

CICS supports the assembler user exit IBMFXITA only for compatibility; however, it is not recommended. Instead you should convert to CEEBXITA, which is supplied by Language Environment. Also, Language Environment does not supply a default IBMFXITA.

If your application requires IBMFXITA, you must:

- Provide IBMFXITA when you install the product. This affects every OS PL/I Version 2 transaction in the CICS region that has not been relinked to run with Language Environment for MVS & VM.
- Relink your application using the linkage editor INCLUDE statement to include the desired IBMFXITA with IBMRSAP or the IBM-supplied default CEEBXITA gets control of every OS PL/I Version 2 transaction in the CICS region.

If you relink your OS PL/I applications with Language Environment, you must use CEEBINT. IBMBINT is not supported for any relinked OS PL/I applications under CICS.

Macro-Level Interface

The CICS macro-level interface is not supported.

Relinking CICS Applications

When you relink OS PL/I object modules (see Chapter 4, “Object and Load Module Considerations” on page 32 for description of object-module support under Language Environment for MVS & VM) with Language Environment for MVS & VM, you must use the following linkage-editor statements:

```
INCLUDE  SYSLIB(CEESTART)
INCLUDE  SYSLIB(CEESG010)
INCLUDE  SYSLIB(DFHELII)
REPLACE  PLISTART
CHANGE   PLIMAIN(CEEMAIN)
INCLUDE  objlib(objmod)
ORDER    CEESTART
ENTRY    CEESTART
NAME     loadmod(R)
```

Where:

objlib represents the PDS that contains the object modules
objmod represents the name of the object module
loadmod represents the name of the resultant load module

The INCLUDE of the object module must occur immediately after the CHANGE statement. Also, the object module of the main procedure must be included before any object modules of subroutines. This was not required for OS PL/I.

SYSTEM(CICS) Compile-Time Option

If you compile with SYSTEM(CICS) compile-time option, PL/I enforces the OPTIONS(BYVALUE) procedure option for MAIN procedures.
OPTIONS(BYVALUE) is the default. If you specify OPTIONS(BYADDR), the compiler diagnoses it as an error and applies OPTIONS(BYVALUE) instead.

FETCHing a PL/I MAIN Procedure

CICS does not support PL/I FETCHing a PL/I MAIN procedure.

STACK Run-Time Option

Language Environment supports PL/I for MVS & VM applications that use the run-time option STACK(,,ANY). Language Environment also supports STACK(,,ANY) for OS PL/I applications that have been relinked with Language Environment for MVS & VM as long as the applications meet the following conditions:

- Application does not contain any edited stream I/O (for example, EDIT was not used in a PUT statement)
- Application specifies AMODE(31).

Run-Time Output

Run-time output is now transmitted to a CICS transient data queue CESE. Language Environment ignores the MSGFILE option under CICS. Figure 2 shows format of the output data queue.

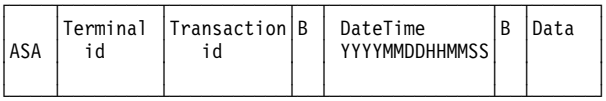

Figure 2. CESE Output Data Queue

In addition, PL/I transient queues CPLI and CPLD are no longer used. As a result, you do not need to specify entries for the CPLI and CPLD in the CICS Destination Control Table (DCT).

Abend Codes Used by PL/I under CICS

The APL*x* abend codes that were issued under OS PL/I Version 2 are no longer issued. Instead, Language Environment-defined abend codes are issued. For more information about Language Environment abend codes, see the *Language Environment for MVS & VM Debugging Guide and Run-Time Messages*.

Shared Library Support

Language Environment provides the same support for OS PL/I Shared Library under CICS as it does under non-CICS. See Chapter 3, “Installation Considerations” on page 24 for details. If you relink your Shared Library using IBMRLSLA, you must add the following to the CSD and add IBMBPSMA and IBMBPSLA to the LPA area.

```
DEFINE PROGRAM(IBMBPSMA) GROUP(CEE) LANGUAGE(ASSEMBLER)
DEFINE PROGRAM(IBMBPSLA) GROUP(CEE) LANGUAGE(ASSEMBLER)
```

If the Shared Library is not supported by Language Environment or you choose not to relink the Shared Library, you must relink your OS PL/I applications that used the Shared Library. In this case, the following actions apply:

- You do not have to specify PLISHRE=YES in the CICS System Initialization Table (SIT).
- You do not have to specify PLI=YES in the SIT.

Linking PL/I for MVS & VM Applications

You are no longer required to take special actions when you link a PL/I for MVS & VM object module under CICS. The CEESTART CSECT is the entry point for programs compiled with OPTIONS(MAIN) or OPTIONS(FETCHABLE). However, if a subroutine that was not compiled with OPTIONS(FETCHABLE) is FETCHed or called, you must code the linkage editor ENTRY statement so that it nominates the actual entry point.

IMS Considerations

Language Environment provides IMS the same level of OS PL/I object and load module support as for non-IMS. See Chapter 4, “Object and Load Module Considerations” on page 32 for details.

Interfaces to IMS

Language Environment supports the PLITDLI, ASMTDLI, and EXEC DLI interfaces from a PL/I routine. It also supports CEETDLI interface from a PL/I for MVS & VM routine running under IMS/ESA Version 4.

Under Language Environment, CEETDLI is the recommended interface. CEETDLI supports calls that use an Application Interface Block (AIB) or a Program Communication Block (PCB). CEETDLI is available under IMS/ESA Version 4. For more information about AIB and a complete description of the CEETDLI interface, see *IMS/ESA Version 4 Application Programming Guide*.

Note that if you recompile your PL/I routine with PL/I for MVS & VM and you want to replace PLITDLI with CEETDLI, you must replace the parameters in the CALL statement with the actual blocks, instead of the pointer to the blocks as required in the CALL PLITDLI statement.

SYSTEM(IMS) Compile-Time Option

The SYSTEM(IMS) option, became available in OS PL/I Version 2, was supported for the PL/I IMS applications only. The main procedure of an IMS application must use the POINTER data type for its parameters.

If you recompile your main procedure with PL/I for MVS & VM, the object module assumes that the parameters are passed as BYVALUE. Language Environment converts the parameters to the BYVALUE style for you if necessary. Therefore, the parameters are always passed correctly. If you specify OPTIONS(BYADDR) when you recompile your main procedure with PL/I for MVS & VM, you receives an error message and the compiler applies BYVALUE instead.

PLICALLA Support in IMS

The OS PL/I PLICALLA entry point is supported under Language Environment. However, it is **not** a recommended interface for IMS. Instead you should use the SYSTEM(IMS) compile-time option and the PLISTART or CEESTART entry point.

Language Environment provides the same support for OS PL/I PLICALLA applications. However, if you recompile your main load module with PL/I for MVS & VM and want to continue to use PLICALLA, you must follow additional rules. See “PLICALLA Considerations” on page 4 for details.

PSB Language Options Supported

Language Environment for MVS & VM supports PL/I applications with the following PSBGEN LANG options in the supported releases of IMS:

IMS/ESA Version 4

Table 16 shows support for PSB LANG options in IMS/ESA Version 4.

Table 16. PSB LANG Options for IMS/ESA Version 4 Release 1

SYSTEM option	Entry point	LANG=
IMS	CEESTART, PLISTART	PLI or other values except PASCAL
IMS	PLICALLA[1]	PLI
Omitted	CEESTART, PLISTART	Illegal
Omitted	PLICALLA[1]	PLI

Note: [1]Supported only for compatibility.

IMS/ESA Version 3 Re1ease 1

Table 17 shows support for PSB LANG options in IMS/ESA Version 3 Release 1.

Table 17. PSB LANG Options for IMS/ESA Version 3 Release 1

SYSTEM option	Entry point	LANG=
IMS	CEESTART, PLISTART	PLI
IMS	PLICALLA[1]	PLI
Omitted	CEESTART, PLISTART	Illegal
Omitted	PLICALLA[1]	PLI

Note: [1]Supported only for compatibility.

Assembler Driving a PL/I Transaction

If an assembler program is driving a transaction program written in PL/I, assuming the PSBGEN LANG= option remains unchanged, the following considerations apply:

- If you recompile the PL/I main program with PL/I for MVS & VM, you must use the SYSTEM(MVS) compile-time option. In this case, no changes to the assembler program are required.
- If you do not recompile the main program with PL/I for MVS & VM, the parameter list format passed from the assembler driver remains unchanged.

Storage Usage Considerations

With IMS/ESA Version 3 Release 1, the parameters passed to the IMS interfaces are no longer restricted to the area below the 16M line. The parameters will most likely be placed above the 16M line if you use the following methods:

- Use the ANYWHERE suboption of the HEAP run-time option; it will apply to variables with the CONTROLLED or BASED attribute because their storage is obtained from the heap.
- Use the ANYWHERE suboption of the STACK run-time option. If you relink your OS PL/I application with Language Environment and your application does not use any edited stream I/O, or you recompile your application with PL/I for

MVS & VM, you can use STACK(,,ANYWHERE) if your application is AMODE(31). In this case, the variables in automatic storage are placed above the 16M line.

- Place parameters in static storage and make sure the load module attribute used is RMODE(ANY).

Coordinated Condition Handling under IMS

Language Environment and IMS condition handling is coordinated, meaning that if a program interrupt or abend occurs when you application is running in an IMS environment, the Language Environment condition manager is informed whether the problem occurred in your application or in IMS. If the problem occurs in IMS, Language Environment, as well as any invoked HLL-specific condition handler, percolates the condition back to IMS.

With Language Environment run-time option TRAP(ON), Language Environment continues to support coordinated condition handling for the PLITDLI and ASMTDLI interface invoked from a PL/I routine.

With IMS/ESA Version 3 with PTF UN4928 or IMS/ESA Version 4, Language Environment also supports the coordinated condition handling for CEETDLI, CTDLI from a C routine, CBLTDLI from a COBOL program, AIBTDLI from a PL/I program, and ASMTDLI from a non-PL/I program.

Note that if a program interrupt or abend occurs in your application outside of IMS, or if a software condition of severity 2 or greater is raised outside of IMS, the Language Environment condition manager takes normal condition handling actions described in Language Environment for MVS & VM Programming Guide. In this case, in order to give IMS a chance to do database rollback, you must do one of the following:

- Resolve the error completely so that you application can continue.
- Issue a rollback call to IMS, and then terminate the application.
- Make sure that the application terminates abnormally by using the ABTERMENC(ABEND) run-time option to transform all abnormal terminations into system abends in order to cause IMS rollbacks.
- Make sure that the application terminates abnormally by providing a modified assembler user exit (CEEBXITA) that transform all abnormal terminations into system abends in order to cause IMS rollbacks.

  The assembler user exit you provide should check the return code and reason code or the CEEAUE_ABTERM bit, and requests an abend by setting the CEEAUE_ABND flag to ON, if appropriate. See Language Environment for MVS & VM Programming Guide for details.

Performance Enhancement with Library Retention(LRR)

If you use LRR, you can get an improvement in performance. See “Improving CPU Utilization” on page 21 for details.

DB2 Considerations

There are no special considerations for using DB2 other than the considerations described in “IMS Considerations” on page 46.

Chapter 8. OS PL/I Coexistence with Language Environment

This chapter discusses how you can run your OS PL/I applications under either OS PL/I or Language Environment. This coexistence gives you the flexibility to migrate your OS PL/I applications to PL/I for MVS & VM and/or Language Environment gradually. It's important that you understand how Language Environment supports OS PL/I object and load modules before you consider the coexistence. For rules and information on Language Environment's support of OS PL/I object and load modules, see Chapter 4, “Object and Load Module Considerations” on page 32.

This chapter discusses the following topics:

- Coexistence under MVS non-CICS
- Coexistence under MVS CICS
- Coexistence under VM.

Coexistence under MVS non-CICS

Under MVS, Language Environment can coexist with the OS PL/I library in the same SMP zone. This enables you to have the Language Environment and OS PL/I library in your environment at the same time and allows you to use either run-time by specifying each library in a certain order in your JCL. Which run-time is used depends on the sequence in which they appear in your JCL and what type of application you are running.

The following library search order rules apply to non-CICS applications and are valid only for pure PL/I applications with the same type of load modules.

Table 18. OS PL/I and Language Environment Coexistence Rules for non-CICS Environment

Type of load module	Search sequence	Run-time used
OS PL/I load module, no Shared Library	1. OS PL/I 2. Language Environment	OS PL/I
	1. Language Environment 2. OS PL/I	Language Environment
OS PL/I load module, with OS PL/I Shared Library	1. OS PL/I 2. Language Environment	OS PL/I
	1. Language Environment 2. OS PL/I	Not supported
OS PL/I load module, with replaced Shared Library	1. OS PL/I 2. Language Environment	Not supported
	1. Language Environment 2. OS PL/I	Language Environment
OS PL/I object module, linked with Language Environment	1. OS PL/I 2. Language Environment	Language Environment
	1. Language Environment 2. OS PL/I	Language Environment
PL/I for MVS & VM	1. OS PL/I 2. Language Environment	Language Environment
	1. Language Environment 2. OS PL/I	Language Environment

If a pure PL/I application contains a mixture of PL/I load modules, the OS PL/I library and Language Environment library must appear in such order that only one run-time environment is used by all load modules in the application. For example, if a PL/I application contains a mixture of any of the following load modules:

- OS PL/I load module, no Shared Library
- OS PL/I object module, linked with Language Environment
- PL/I for MVS & VM load module

Language Environment must always be placed before the OS PL/I library in the search order.

If your application contains ILC, the search order of Language Environment for MVS & VM must be correct for all ILC programs in the application.

Coexistence under MVS CICS

CICS allows multiple language run-time environments to coexist in the same CICS region. Therefore, you can bring up the CICS region with both OS PL/I and Language Environment. However, if Language Environment exists and the PL/I in Language Environment is enabled, Language Environment is used for all PL/I transactions, including OS PL/I. It is only when Language Environment does not exist or the PL/I component in Language Environment is not enabled, that the OS PL/I environment is used for OS PL/I transactions. In this case, you cannot run PL/I for MVS & VM transactions because it requires Language Environment with the PL/I component enabled. The following table summarizes the coexistence support for OS PL/I transactions:

Table 19. Summary of Coexistence Support under MVS CICS

PPT	OS PL/I	Language Environment	Environment Used
PLI	Yes	Yes, with PL/I enabled	Language Environment
PLI	Yes	Yes, without PL/I enabled	OS PL/I
PLI	No	Yes, with PL/I enabled	Language Environment
PLI	No	Yes, without PL/I enabled	Abend APCL
LE370	Yes	Yes, with PL/I enabled	Language Environment
LE370	Yes	Yes, without PL/I enabled	Abend APCL
LE370	No	Yes, with PL/I enabled	Language Environment
LE370	No	Yes, without PL/I enabled	Abend APCL

The following shows how you can disable the PL/I component from Language Environment:

- If Language Environment is not installed, do the following:
  - Delete CEEEV010 module from CEECCSD located in Language Environment SCEESAMP
  - Run the DFHCSDUP utility with CEECCSD
- If Language Environment is installed, do the following:
  - CEDS DELETE PROGRAM(CEEV010), bring down CICS, and start CICS cold

You must concatenate OS PL/I in front of Language Environment SCEERUN in DFHRPL in order to use the OS PL/I environment.

Coexistence under VM

Under VM, OS PL/I and Language Environment must be installed on separate minidisks. Because Language Environment contains relocatable load modules that are given control by VM before any loadlib or txtlib, you must access only one minidisk at any one time, depending on which run-time environment you use.

If any portion of Language Environment is installed in nuxleus extension or named saved segments (NSS), you must always access the minidisk for Language Environment, that is, you cannot use the OS PL/I environment.

Chapter 9. Migration Aids

Language Environment provides you several sample JCL and EXEC located in the sample library SCEESAMP that can help you to migrate to Language Environment. The migration aids available with Language Environment are:

- OS PL/I library routine replacement tool
- OS PL/I Version 1 Release 5.1 main load module ZAP
- OS PL/I Shared Library replacement tool

OS PL/I Version 2 Release 3 also provides you a migration aid that helps you to migrate your PL/I-COBOL ILC applications and PLISRTx applications. The migration aid is called:

- OS PL/I object module relinking tool - APAR PN69803

You can use the library CSECT names in the load module to identify the functions that are used in the module. A list of library CSECT names are provided at the end of this chapter that you can use to identify load modules that you need to take some actions when you migrate to Language Environment if the load module contains a function that Language Environment does not support.

OS PL/I Library Routine Replacement Tool

Language Environment does not support OS PL/I Version 1 Release 3.0 - 5.0 load modules. For those load modules, you can relink the object modules directly with Language Environment, or you can replace the library routines in the load module with the Language Environment stubs. See Chapter 4, “Object and Load Module Considerations” on page 32 for detailed OS PL/I object and load module support.

Language Environment provides two samples located in SCEESAMP that you can use to replace the library routines in your OS PL/I Version 1 Release 3.0 - 5.1 and Version 2 load modules with corresponding Language Environment stubs. These two samples simply contain a list of linkage editor REPLACE control statements to replace each library routine in your load module with the corresponding stub in Language Environment.

- IBMWRLK is for MVS non-CICS and VM. You can use it to replace OS PL/I V1R3.0 - V1R5.1 and V2 load modules, both multitasking and nonmultitasking. Note that it contains a CHANGE statement to rename the OS PL/I HLL user exit IBMBINT to CEEBINT.

  Under VM, if the load module was created with a LKED command (the load module resides in a VM LOADLIB), you can use IBMWRLK to replace the load module.

  If the load module was created with the GENMOD command, you cannot use IBMWRLK to replace the load module.

- IBMWRLKC is for CICS. You can use it to replace OS PL/I V1R3.0 - V1R5.1 and V2 load modules. Note that it contains a CHANGE statement to rename the OS PL/I HLL user exit IBMBINT to CEEBINT and PLIMAIN to CEEMAIN. It also contains INCLUDE statements to ensure the load module works under CICS.

The CICS macro language is not supported.

The following MVS JCL example shows the replacement of run-time library routines from a user load module while retaining the user object module. In the example, MYPDS.LOAD is the data-set name of a load module library that contains the load module with the name MYLMOD.

```
//RELINK    EXEC PGM=IEWL,PARM='LIST,MAP,XREF,SIZE(3072K,4K)',REGION=5M
//SYSPRINT  DD   SYSOUT=A
//SYSLIB    DD   DSN=CEE.V1R4M0.SCEELKED,DISP=SHR
//SAMPLIB   DD   DSN=CEE.V1R4M0.SCEESAMP,DISP=SHR
//SYSUT1    DD   UNIT=SYSDA,SPACE=(1024,(200,200))
//SYSLMOD   DD   DSN=MYPDS.LOAD,DISP=OLD
//SYSLIN    DD   *
  INCLUDE   SAMPLIB(IBMWRLK)
  INCLUDE   SYSLMOD(MYLMOD)
  NAME      MYLMOD(R)
```

If you replace a load module under CICS, the CICS SDFHLOAD dataset must be specified in the SYSLIB.

The following VM example shows the replacement of run-time library modules from the user load module MYLMOD which is a member of the VM LOADLIB with the file name MYLLIB while retaining the user object module. In this example, there must be a file with a file name of MYTEXT and a file type of TEXT which contains the following linkage editor control statements:

```
   INCLUDE  SAMPLIB(IBMWRLK)
   INCLUDE  MYLLIB(MYLMOD)
   NAME     MYLMOD(R)
```

With MYLLIB LOADLIB on the A disk, the following commands link edit the member MYLMOD so that it can run with Language Environment for MVS & VM:

```
FILEDEF  MYLLIB DISK MYLLIB   LOADLIB A (RECFM U
FILEDEF  SAMPLIB DISK SCEESAMP *
LKED     MYTEXT (LIST MAP XREF LIBE MYLLIB NAME MYLMOD SIZE 9999K 100K NOTERM
FILEDEF  * CLEAR
```

If the FETCH statement is used, both load modules must use the Shared Library with equivalent dummy transfer vectors, or neither can use the Shared Library. This existing rule is described in “Using the Shared Library” in *OS PL/I Programming Guide*

OS PL/I Version 1 Release 5.1 Main Load Module ZAP

Language Environment supports OS PL/I Version 1 Release 5.1 main load module with a restriction that if the main load module is for MVS non-Shared Library, non-CICS and nonmultitasking, or VM, it must be ZAPped with Language Environment-provided sample ZAP IBMRZAPM for MVS (or IBMRZAPV for VM) first. Only the ZAPped main load module is supported by Language Environment.

If you choose not to ZAP your main load module, you can read “OS PL/I Library Routine Replacement Tool” on page 53 to understand what else you can do. You can also recompile your application with PL/I for MVS & VM or OS PL/I Version 2. See Chapter 4, “Object and Load Module Considerations” on page 32 to understand how Language Environment supports OS PL/I object and load modules.

The sample ZAP is located in Language Environment SCEESAMP. It is also available in the IBM Support Center for customers who do not have Language Environment but want to prepare to migrate to Language Environment.

- IBMRZAPM for MVS non-Shared Library, non-CICS, nonmultitasking

  The ZAPped main load module continues to run under OS PL/I Version 1 Release 5.1 and Version 2. If the main load module contains the OS PL/I fast initialization and termination feature, the feature is supported also. However, in this case, the ZAPped main load module always dynamically loads the OS PL/I run-time initialization routine IBMBPIIA once. The dynamically loaded IBMBPIIA is not deleted until the task terminates. Loading IBMBPIIA dynamically once while you are using OS PL/I fast initialization and termination feature might affect the performance of your application. If you put IBMBPIIA in LPA, the performance effect can be minimized.

  The ZAPped main load module is supported by Language Environment. However, if the load module contains the OS PL/I fast initialization and termination feature, the feature is not supported by Language Environment. Language Environment always dynamically loads the initialization and termination routines. If you put the Language Environment library routines and CEEBLIIA in LPA(E) as recommended in *Language Environment for MVS & VM Installation and Customization under MVS*, the performance effect can be minimized.

- IBMRZAPV for VM

  The ZAPped main load module is NOT supported under OS PL/I Version 1 Release 5.1 or Version 2. It's supported under Language Environment only.

The instructions for how to use the ZAP is described in IBMRZAPM and IBMRZAPV.

The OS PL/I Version 1 Release 5.1 load module ZAP is also available from the IBM Support Center for the customers who do not have Language Environment but want to prepare for migration.

OS PL/I Shared Library Replacement Tool

The library module in the OS PL/I Version 1 Release 5.1 and Version 2 Shared Library must be replaced with Language Environment stubs in order to support OS PL/I Version 1 Release 5.1 and Version 2 load modules that use the shared library.

Language Environment provides two sample JCL located in SCEESAMP to help you to replace the Shared Library:

- IBMRLSLA for OS PL/I Version 1 Release 5.1 MVS CICS or multitasking and OS PL/I Version 2 Shared Library
- IBMRLSLB for OS PL/I Version 1 Release 5.1 MVS non-CICS nonmultitasking Shared Library

You must understand how Language Environment supports OS PL/I Shared Library before you use the JCL. See “Considerations for relinking the Shared Library” on page 30 and “Load Module Using the Shared Library” on page 33 for detailed OS PL/I Shared Library and load module support.

OS PL/I Object Module Relinking Tool - APARs PN69803

OS PL/I Version 2 Release 3 provides APAR PN69803 help you migrate your PL/I-COBOL ILC applications and PLISRTx applications.

Language Environment does not support the OS PL/I-COBOL ILC applications. The OS PL/I object module in the PL/I-COBOL ILC application must be relinked. See “Differences in Interlanguage Communication Support” on page 19 for the ILC support under Language Environment. However, if your OS PL/I object module in the PL/I-COBOL ILC application is relinked with PN69803 the resultant load module is supported by Language Environment. Therefore, you don't need to relink your OS PL/I object module in the PL/I-COBOL ILC application with Language Environment. PN69803 provide you the flexibility that you can prepare the PL/I-COBOL ILC relinking while you are using OS PL/I Version 2 Release 3. When you complete the relinking, you can switch to Language Environment whenever you are ready.

OS PL/I applications that use PLISRTx is supported by Language Environment. We recommend that you relink the application. See “Differences in PLISRTx Support” on page 10 for the reasons. The recommended relinking can be done either with Language Environment or with PN69803 on OS PL/I Version 2 Release 3. Either way will give your load module the benefits of exploiting the Language Environment DFSORT interface support.

For the PL/I-COBOL ILC applications, before you relink them with PN47774, you must apply the following PL/I-COBOL ILC APARs to PL/I and COBOL first:

- OS PL/I V2R3 common library: PN36844
- VS COBOL II V1R3.0 library: PN13459
- VS COBOL II V1R3.1 library: PN04721
- VS COBOL II V1R3.2 library: PN09732

Note: VS COBOL II V1R4.0 has the above COBOL APARs in its base code.

If you have not applied the above APARs, PN69803 will not work. Note that the above APARs are not required if your application does not contain PL/I-COBOL ILC.

You must be aware that even though your PL/I-COBOL ILC applications are relinked with PN69803 they might still be required to link with Language Environment if they contain a function described in this book or in *COBOL for MVS & VM Migration Guide*. For example, if the application contains any COBOL NORES or the load module contains an OS PL/I object module which is not supported by Language Environment. In the later case, you must recompile your OS PL/I object module with PL/I for MVS & VM or OS PL/I Version 2.

Identifying Functions Used in an OS PL/I Load Module

You can use the library CSECT names in your OS PL/I load module to identify certain functions that are used in the module. This can help you to identify load modules that you need to take some actions when you migrate to Language Environment if the load module contains a functions that is not supported by Language Environment.

Note that some features that are not supported or require relink by Language Environment cannot be identified by searching the library CSECT names in the load module.

Table 20. Using CSECTs to Identify Problem Load Modules

CSECT name	Module type
IBMBILC1	ILC load module
IBMBIEC1	ILC module in which PL/I calls COBOL
IBMBIEF1	ILC module in which PL/I calls FORTRAN
IBMBIEP1	ILC module in which COBOL or FORTRAN calls PL/I
IBMTPIR1	Multitasking module
IBMBKST1	PLISRTx module
IBMBPII1	Version 1 fast initialization and termination
IBMBPOPT	PLIXOPT string options table
IBMBDIM1	PLITEST invocation module
IBMBPSR1	Shared Library addressing module for nonmultitasking
IBMTPSR1	Shared library addressing module for multitasking

Bibliography

PL/I for MVS & VM Publications

- *Licensed Program Specifications*, GC26-3116
- *Installation and Customization under MVS*, SC26-3119
- *Compiler and Run-Time Migration Guide*, SC26-3118
- *Programming Guide*, SC26-3113
- *Language Reference*, SC26-3114
- *Reference Summary*, SX26-3821
- *Compile-Time Messages and Codes*, SC26-3229
- *Diagnosis Guide*, SC26-3149

Language Environment for MVS & VM Publications

- *Fact Sheet*, GC26-4785
- *Concepts Guide*, GC26-4786
- *Licensed Program Specifications*, GC26-4774
- *Installation and Customization under MVS*, SC26-4817
- *Programming Guide*, SC26-4818
- *Programming Reference*, SC26-3312
- *Debugging Guide and Run-Time Messages*, SC26-4829
- *Writing Interlanguage Communication Applications*, SC26-8351
- *Run-Time Migration Guide*, SC26-8232
- *Master Index*,SC26-3427

PL/I for OS/2 Publications

- *Programming Guide*, SC26-8001
- *Language Reference*, SC26-8003
- *Reference Summary*, SX26-3832
- *Built-In Functions*, SC26-8089
- *Installation*, SX26-3833
- *Messages and Codes*, SC26-8002
- *License Information*, GC26-8004
- *WorkFrame/2 Guide*, SC26-8000

CoOperative Development Environment/370

- *Fact Sheet*, GC09-1861
- *General Information*, GC09-2048
- *Installation*, SC09-1624
- *Licensed Program Specifications*, GC09-1898
- *User's Guide and Reference*, SC09-1623
- *Using CODE/370 with VS COBOL II and OS PL/I*, SC09-1862
- *Self-Study Guide*, SC09-2047

IBM Debug Tool

- *User's Guide and Reference*, SC09-2137

Softcopy Publications

Online publications are distributed on CD-ROMs and can be ordered from Mechanicsburg through your IBM representative. PL/I books are distributed on the following collection kit:

- *Application Development Collection Kit*, SK2T-1237

Other Books You Might Need

CICS/ESA

- *Application Programming Reference*, SC33-0676

DFSORT

- *Application Programming Guide*, SC33-4035

DFSORT/CMS

- *User's Guide*, SC26-4361

IMS

- *IMS/ESA V4 Application Programming: Database Manager*, SC26-3058
- *IMS/ESA V4 Application Programming: Design Guide*, SC26-3066
- *IMS/ESA V4 Application Programming: Transaction Manager*, SC26-3063
- *IMS/ESA V4 Application Programming: EXEC DL/I Commands for CICS and IMS*, SC26-3062

MVS/DFP

- *Access Method Services*, SC26-4562

MVS/ESA 4.3 MVS Support for OpenEdition Services Feature

- *Introducing OpenEdition MVS*, GC23-3010
- *OpenEdition MVS POSIX.1 Conformance Document*, GC23-3011
- *OpenEdition MVS User's Guide*, SC23-3013
- *OpenEdition MVS Command Reference*, SC23-3014

MVS/ESA

- *JCL User's Guide*, GC28-1473
- *JCL Reference*, GC28-1479
- *System Generation,* CG28-1825
- *System Programming Library: Initialization and Tuning Guide,* GC28-1451
- *System Messages Volume 1*, GC28-1480
- *System Messages Volume 2*, GC28-1481
- *System Messages Volume 3*, GC28-1482
- *System Messages Volume 4*, GC28-1483
- *System Messages Volume 5*, GC28-1484

OS/VS2

- *TSO Command Language Reference*, GC28-0646
- *TSO Terminal User's Guide*, GC28-0645
- *Job Control Language*, GC28-0692
- *Message Library: VS2 System Codes*, GC38-1008

SMP/E

- *User's Guide*, SC28-1302
- *DBIPO Dialogs User's Guide*, SC23-0538
- *Reference*, SC28-1107
- *Reference Summary*, SX22-0006

TCAM

- *ACF TCAM Application Programmer's Guide*, SC30-3233
- *OS/VS TCAM Concepts and Applications*, GC30-2049

TSO/E

- *Command Reference,* SC28-1881

VM/ESA

- *CMS User's Guide,* SC24-5460
- *CMS Command Reference,* SC24-5461
- *CMS Application Development Guide*, SC24-5450
- *XEDIT User's Guide,* SC24-5463
- *XEDIT Command and Macro Reference,* SC24-5464
- *CP Command and Utility Reference,* SC24-5519
- *Installation,* SC24-5526
- *Service Guide,* SC24-5527
- *System Messages and Codes*, SC24-5529.

Index

A

C

D

E

F

G

H

I

L

M

N

O

P

R

S

T

U

V

X

Z